Real-time multi-platform pedestrian detection in a heavy duty driver assistance system

Songlin Piao[1], Lisa Kiekbusch[1], Daniel Schmidt[1], Karsten Berns[1], Daniel Hering[2], Stefan Wirtz[2], Nils Hering[2] and Jürgen Weiland[2]

[1] Technische Universität Kaiserslautern, Gottlieb-Daimler-Straße, 67663 Kaiserslautern
[2] Motec Entwicklungszentrum für Nutzfahrzeugassistenzsysteme, Universitätsstr. 3, 56070 Koblenz

Abstract. Reliable and fast pedestrian detection in mobile systems is an important and increasing field of research. The main goal is to save lives of pedestrians in as many situations and environments as possible. Most of the existing reliable pedestrian detection algorithms run on high performance PCs with intense CPU and GPU usage and are therefore not suited for mobile embedded systems. We present a heavy duty multi-platform mobile system running a real-time pedestrian detection. Our approach is to run each part of our software on the hardware part which is suited best for solving the problem. This distribution of the software to the several hardware parts is work in progress, but first experiments with the basic algorithm already show promising results.

1 Introduction

The raised demands for traffic safety make reliable pedestrian detection on mobile systems necessary. Besides the quality requirement, an acceptable price and low power consumption are also important for advanced driver assistance systems (ADAS) in the commercial vehicle domain. Distributed computing could be the key to achieve a real-time pedestrian detection with energy-efficient low-cost devices. To match these criteria, we present a distributed system consisting of three hardware platforms: (i) a set of intelligent cameras, (ii) a fusion-box and (iii) an embedded PC. With this setup we will be able to achieve a catalog price below $10,000$€ for the overall system.

We selected four cameras with a wide-angle lens in order to monitor as much space as possible around the vehicle. Our fusion-box uses a System on Chip (SoC) running a customized Linux. This system combines the advantages of parallel-computing using the FPGA and processing of floating point operations with an ARM processor. For the embedded PC the Generic Control Box[1] is used to run control software.

Implementing an efficient pedestrian detection algorithm plays an important role in the proposed system. Most of the state-of-the-art algorithms cannot be directly used in our system due to their computational effort. We relaxed the real-time issue by building descriptors using Local Binary Pattern (LBP) and applying linear support vector machines (SVM) and integral images, similar to the approach presented in Piao et al. [14]. We applied the detection algorithm on each camera stream and combined the results

[1] www.robotmakers.de

with additional information gained from camera calibration, camera height and intrinsic camera parameters. The final results include the type and orientation of the detected objects, the estimated distance and the probability of being an interesting object. The results are then serialized and streamed to the embedded PC, where the data is saved for further application tasks. With the aim of improving the processing speed, we have identified the time consuming tasks and started moving them from the CPU to the FPGA.

We perform test-drives on a truck under several weather conditions and times of day to measure the quality of the system. First experiments make us confident that the results provide the needed precision and robustness for reliable pedestrian detection. The remainder of this paper is organized as follows. Section 2 summarizes the related work in pedestrian detection area. In Section 3 the proposed methodology and algorithms will be discussed in more detail. The application examples and experimental results will be described in section 4 and section 5 gives the conclusion and future work.

2 Related Work

In order to achieve a high detection rate with only few false positives, researchers have been trying to find effective ways to solve the problem [4]. In [2] and [6], state-of-the-art performances on standard data sets are discussed. Both detectors were trained with AdaBoost and the feature pool was generated from multiple channels of the original input image. The authors of [16] proposed the histogram of oriented gradient (HOG) based cascaded classifier and applied it to a commercial product. Locally normalized HOG descriptors in a dense overlapping grid and trained final classifiers with linear SVM are utilized in [5]. The proposed method becomes a baseline of many other human detectors [7]. In [8], an object detection system being based on a mixture of multi-scale deformable part models and showing best performance in PASCAL object detection challenges is proposed. [13] took the advantage of deep learning to jointly learn feature extraction, deformation handling, occlusion handling and classification. In [9], an on-board pedestrian detection system, in which Haar wavelets and edge orientation histograms are used to classify the incoming ROIs, is presented.

In addition, a lot of efforts have been put to get optimized balance between speed and accuracy. A lot of research is done using the graphics processing unit (GPU) to speed up computation time. Nevertheless, these kinds of implementations were still too computationally intensive for a practical system which may be operated by robotic systems, mobile devices, or intelligent vehicles. The authors of [12] proposed a simple yet powerful branch and bound scheme that allows efficient maximization of a large class of quality functions over all possible subimages. In [17], a real-time human detection algorithm using contour cues with CENTRIST descriptor is proposed. [6] proposed the way of building finely sampled feature pyramids at a fraction of the cost without losing performance. Others achieved processing speed of over 100 fps through direct stixel world computation without depth map [3]. In [1], an architecture for pedestrian detection where scaling and gradient computation are implemented on the FPGA is presented. We used the description of their implementation as a basis for ours.

3 Pedestrian Detection System Design

3.1 System Overview

Considering the requirements from quality, price, and power consumption in the commercial vehicle domain we developed a hardware system suited for distributed computing. The system consists of an intelligent camera, a fusion-box and an embedded PC. Each part of the software is supposed to be running on the part of the hardware which qualifies best for the particular problem. By establishing a fast and reliable communication between each platform we manage to achieve a real-time pedestrian detection with energy-efficient low-cost devices. In this subsection, we will introduce our hardware setup in more detail and give an overview of the whole system.

On each side of the vehicle, a camera with 180° horizontal field of view is mounted to monitor as much area as possible. The camera achieves a resolution of 720x480 pixels and uses the encoding system NTSC for data transfer in the current state. In the final phase it will be replaced by an Ethernet camera which will give us advantages concerning image transmission and a higher resolution of at least 720x576 pixels. The cameras are mounted in 170cm height which corresponds approximately to the height of humans. The viewing direction of the cameras is parallel to the ground. The mounting position and viewing direction of the cameras are chosen considering the performance of the currently implemented detection algorithm.

The fusion-box mainly consists of an ARM Cortex-A9 CPU and an Altera FPGA. A Yocto based customized Linux is the current operating system. To achieve a fast and reliable communication between these both parts, a communication IP core with a Linux driver was developed. This core allows communication and data transfer in both directions. Due to this infrastructure we are able to port time-consuming but parallelizable tasks from the CPU to the FPGA. Additionally, the fusion-box integrates radar or ultrasonic sensor data. The data is transferred by CAN bus which is a standard message-based protocol in vehicles. Our fusion-box merges the received data with the corresponding camera images.

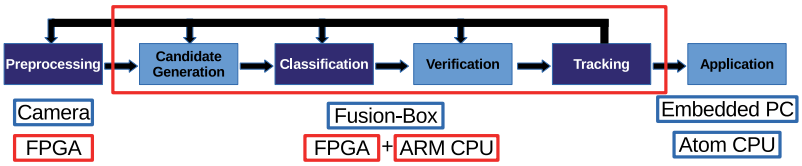


Fig. 1. Processing Chain of the Pedestrian Detection System

From the algorithm point of view, six modules should be implemented for the final product. Fig. 1 shows the general flow path through the designed pedestrian detection system. The **Preprocessing** module includes image rectification, scaling and channel computation. The **Candidate Generation** module generates possible image areas which may contain interesting objects. The **Classification** module is used for judging

if the extracted region contains interesting objects. The **Verification** module validates the detected results based on other conditions, e.g. geometry constraints. This helps to reduce unnecessary false detections and improve system robustness. The **Tracking** module is used for remedying the possibly missed targets from the detector part. The latter four modules contained inside the red rectangle in Fig. 1 are designed to run on the fusion-board. The last component, the **Application** module, is responsible for collecting detected results from previous modules and providing them for the user interface.

The Generic Control Box equipped with an Intel Atom CPU runs the **Application** module. The module collects not only the the results from vision sensors but also the results from other types of sensors like those of the laser scanner. The **Application** module stores the data for further processing, e.g. combination with data from other sensors and generation of application-oriented views for vehicle drivers. In this paper, we will focus on two aspects: object detection in the **Classification** module and object tracking in the **Tracking** module. The details of the FPGA implementation will be described in section 3.4.

3.2 Classification Module

Fig. 2 shows the general procedure of building descriptors for the human patch. As a preprocessing step, several channel images are generated from source image, then descriptors are built with predefined grid patterns directly on the channel images. Fig. 2(a) is an original RGB color image, Fig. 2(b) is a converted SobelEdge image and Fig. 2(c) shows the 16 by 8 feature extraction grid pattern used in [5] and Fig. 2(d) shows the final HOG descriptor of the given patch. The number of channel images may not be limited to one, as example, the Aggregate Channel Features (ACF) based method [6] utilized ten channels while the HOG based method [5] generated only a single gradient channel image. Generally, if we denote F as a feature model of an object, the definition of F can be written as follows

$$F = F_1 \oplus F_2 \cdots F_{N-1} \oplus F_N \tag{1}$$

Eq. 1 shows that the feature model F consists of N sub-features F_i, where $i = 1, \cdots, N$. More specifically, we can describe the HOG based descriptor as follows

$$f_{HOG} = f_{1,1} \oplus f_{1,2} \cdots f_{15,6} \oplus f_{15,7} \tag{2}$$

The final descriptor of HOG was extracted from 105 super blocks; a super block consists of adjacent 2 by 2 grids as shown in Fig. 2(c). In Eq. 2, $f_{i,j}$ represents the feature model for each super block, where i and j are the row index and column index of the super block. Contour information is further used in the designed system to extract human descriptors, but Local Binary Pattern (LBP) replace the original HOG feature. First, each patch is divided into several grids; second, a LBP based histogram is calculated for every possible super block; at the end, the final descriptor is generated by concatenating all histograms. As the length of each LBP based histogram is 256, the final descriptor length is 256 times the number of total super blocks. The index of each bin in the histogram is represented as $(C_1C_2C_3C_4C_5C_6C_7C_8)_2$ where C_i is set to 1 if the corresponding neighbor pixel value is higher than the current pixel value otherwise it is set to 0.

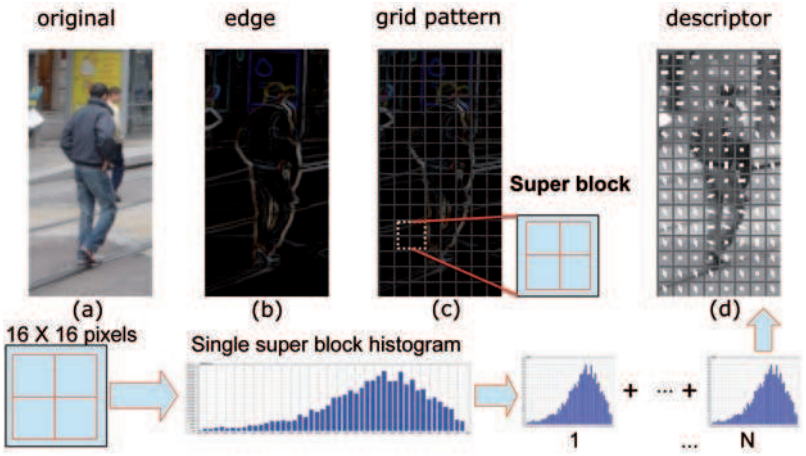


Fig. 2. Descriptor Generation

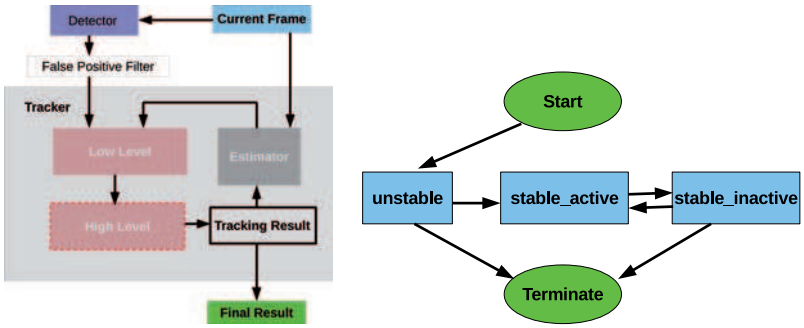
3.3 Tracking Module

Fig. 3(a) shows the structure of designed tracking module. The gray rectangle represents the tracking part in which two levels of data association and an estimator part are included. The data association is solved by the hungarian algorithm [11]. For the estimator, we propose to use the Median Flow tracker [10]. In order to manage the target consistency in a good manner, the framework internally defines three target states: **unstable**, **stable_active** and **stable_inactive**. The **unstable** and **stable_active** trackers will firstly get associated with the incoming detector outputs in the **Low Level**. Later, the **stable_inactive** trackers get associated in the **High Level**. Among these three states, **stable_active** is the only one to be allowed to use the functions from **Estimator**. Fig. 3(b) shows the transition relationship among three different states.

During the matching score calculation, we consider not only a relative distance between a tracker and a detection response, but also the differences in the appearance model and the rectangle size. The appearance model is represented using the color histogram; the Hue and Saturation channels are used for calculating the histogram. The matching score between a tracker tr and a detection output d in each data association level can be computed using Eq. 3. A_{pos}, A_{size} and A_{app} are affinity models based on position, size and appearance, respectively.

$$\begin{aligned} LowLevel(tr,d) &= A_{pos}(tr,d)A_{size}(tr,d)A_{app}(tr,d) \\ HighLevel(tr,d) &= A_{app}(tr,d) \end{aligned} \tag{3}$$

For the high level association, only the appearance model is considered. It makes sense, because there is no predicted rectangle available for the **stable_inactive** trackers. After these two levels in the data association process, for each detection output d, either it



(a) Structure (b) State Transition

Fig. 3. Tracking Module

gets associated with a tracker or it does not get associated with any tracker. For the first case, it helps to update the internal state of the associated tracker; for the second case, it is used for initializing a new **unstable** tracker. For target estimation, only the **stable_active** trackers are allowed to predict possible positions in the next frame. The Median Flow tracker [10] was chosen as the estimator, because it automatically detects the tracking failure and provides reliable results.

3.4 FPGA Integration

We have identified the time consuming tasks and started to move them from the CPU to the camera and the FPGA, respectively. Those tasks include (i) rectification of the image, (ii) Sobel filter and (iii) image scaling. Each of these tasks is very important for a good quality of the pedestrian detection, very time consuming in the CPU computation and highly parallelizeable and therefore suited for a FPGA implementation.

The implementation on the FPGA is currently work in progress but the block diagrams for (i) and (ii) describe the ongoing implementation. Fig. 4 shows the rectification process and Fig. 5 shows our FPGA implementation of the Sobel filter. Our implementation approach is comparable to the one presented in [15]. The module needs to have access to a 3×3 window which is buffered in the line buffer. The actual Sobel filter processes all pixels which corresponding mask values are unequal to zero. The multiplications with 2 are calculated as bit shifts which gives a performance benefit. Image scaling will work quite similar to the process described in Fig. 4 but uses a look up table (LUT) for bilinear interpolation.

4 Application Example

There are two expected application scenarios of the proposed pedestrian detection system: One is garbage trucks in the communal sector and the other is construction machines. The evaluation was conducted from two points of view. Firstly, we evaluated the

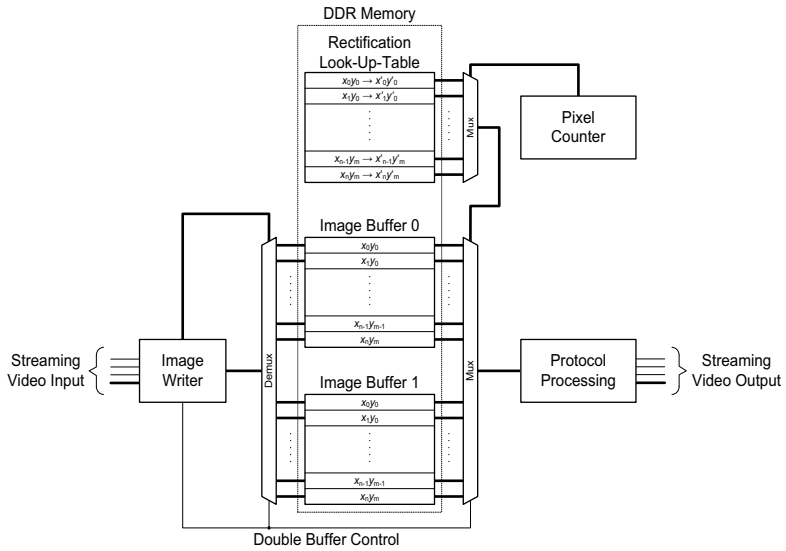


Fig. 4. The rectification process in the FPGA (i)

designed **Classification** and **Tracking** modules on a standard test set of ETH-Bahnhof. Secondly, we tested the current prototype system in real situations by setting up a real test bed including the presented hardware running the algorithm.

Fig. 6 shows the test results on the ETH-Bahnhof dataset. The test sequence consists of 1000 frames taken by a moving camera mounted on a chariot in a crowed street with a 640 x 480 resolution. One can see that with the help of the **Tracking** module, the miss detection rate can be reduced to around 40%, which shows no big difference compared to the state-of-the-art method. In this test, no **Candidate Generation** and **Verification** modules were used, the whole image was fully scanned by the scanning window approach with the scaling factor of 0.8 and scanning step size of 2 pixels. When the scanning window size is 108 by 36, the total number of scanning windows is around 546000. The average computation time per frame including the **Tracking** module for the Intel(R) Core(TM) i5-2500 CPU from 2011 is around 90 ms in release mode. The computation time would decrease to 30 ms per frame when only the **Classification** module is used. The ARM Cortex-A9 CPU on the fusion-box runs about 30 times slower than the above mentioned CPU, which will be counteracted by the described FPGA implementation.

Since the hardware is currently in development stage, not all components could be tested so far in the real testing environment. Especially the usage of FPGA-features described in Sec. 3.4 are not integrated in the test cases. We equipped our test vehicle with four cameras and distinguished three test settings in order to find out the best

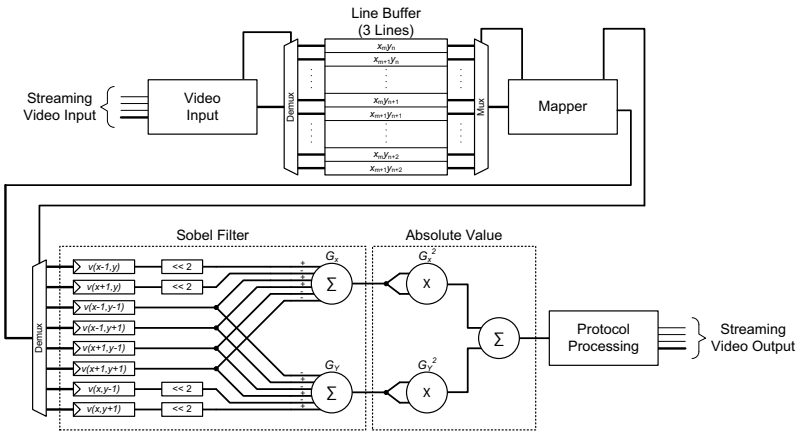


Fig. 5. The Sobel filtering process in the FPGA (ii)

camera properties, test the algorithm on real videos during a refuse collection and an excavation process on a construction site, and test the algorithm on the mobile platform in real-time.

Fig. 7 shows some pictures of the test runs. The results reveiled that the algorithm works well independent of the opening angles of the cameras. Therefore, we decided to use the 180° cameras to get the maximal field of view. Several test situations are passed, e.g. different distances between people and vehicle, groups of people, partial occlusion of people by refuse bins, several weather conditions such as sunshine and drizzle. The test runs provide good results as shown in Fig. 7. First tests with the algorithm running on the mobile platform provide promising results. Up to now, by setting the region of interest in the input image, the system needs 250 ms per frame; the speed will be further boosted through the FPGA implementation (see Sec. 3.4).

5 Conclusion and Future Work

We presented a distributed hardware system running a pedestrian detection. Our system consists of three hardware platforms: an intelligent camera, a fusion-box and an embedded PC. Each part of the hardware is supposed to run different parts of the software system and to be used accordingly to its strengths. The pedestrian detection is already running in a prototypical version on our hardware platform without using the full FPGA acceleration yet. First tests already showed promising results. The soft real-time requirement will be reached with the full FPGA usage. This FPGA acceleration is work in progress and our first step in future work. A more elaborated evaluation of the complete system will be an important next step as well.

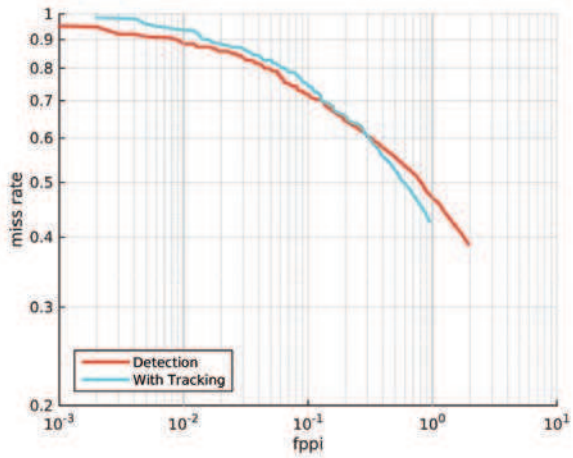


Fig. 6. False positives per image (FPPI) vs. miss-rate curves for the image sequences of the ETH-Bahnhof Dataset.

ACKNOWLEDGMENT

The work at hand was funded from the Federal Ministry of Education and Research (BMBF) under grant agreement number 01|S13027D.

References

1. Sebastian Bauer, Sebastian Kohler, Konrad Doll, and Ulrich Brunsmann. Fpga-gpu architecture for kernel svm pedestrian detection. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, pages 61–68, June 2010.
2. Rodrigo Benenson, Mathias Markus, Tinne Tuytelaars, and Luc Van Gool. Seeking the strongest rigid detector. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3666–3673, June 2013.
3. Rodrigo Benenson, Markus Mathias, Radu Timofte, and Luc J. Van Gool. Pedestrian detection at 100 frames per second. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 2903–2910, 2012.
4. Rodrigo Benenson, Mohamed Omran, Jan Hosang, and Bernt Schiele. Ten years of pedestrian detection, what have we learned? In Lourdes Agapito, Michael M. Bronstein, and Carsten Rother, editors, *Computer Vision - ECCV 2014 Workshops*, volume 8926 of *Lecture Notes in Computer Science*, pages 613–627. Springer International Publishing, 2015.
5. Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In Cordelia Schmid, Stefano Soatto, and Carlo Tomasi, editors, *International Conference on Computer Vision & Pattern Recognition*, volume 2, pages 886–893, INRIA Rhône-Alpes, ZIRST-655, av. de l'Europe, Montbonnot-38334, June 2005.

(a) Front camera (115°) (b) Right camera (90°) (c) Right camera (180°)

Fig. 7. Images of a refuse collection scenario using cameras with different opening angles and mounting positions. Red boxes mark pedestrians identified by the pedestrian detection algorithm.

6. Piotr Dollár, Ron Appel, Serge Belongie, and Pietro Perona. Fast feature pyramids for object detection. *PAMI*, 2014.
7. Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *PAMI*, 34, 2012.
8. Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, September 2010.
9. David Gerónimo, Angel D. Sappa, Daniel Ponsa, and Antonio M. López. 2d-3d-based on-board pedestrian detection system. *Computer Vision and Image Understanding*, 114(5):583 – 595, 2010. Special issue on Intelligent Vision Systems.
10. Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-backward error: Automatic detection of tracking failures. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 2756 –2759, aug. 2010.
11. Harold W. Kuhn and Bryn Yaw. The hungarian method for the assignment problem. *Naval Res. Logist. Quart*, pages 83–97, 1955.
12. Christoph H. Lampert, M.B. Blaschko, and T. Hofmann. Efficient subwindow search: A branch and bound framework for object localization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(12):2129–2142, Dec 2009.
13. Wanli Ouyang and Xiaogang Wang. Joint deep learning for pedestrian detection. In *IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013*, pages 2056–2063, 2013.
14. Songlin Piao and Karsten Berns. Vision based person detection for safe navigation of commercial vehicle. In *Proceedings of the 13the International Conference on Intelligent Autonomous Systems (IAS-13)*, Padova, Italy, July 15-19 2014.
15. Varun Sanduja and Rajeev Patial. Article: Sobel edge detection using parallel architecture based on fpga. *International Journal of Applied Information Systems*, 3(4):20–24, July 2012. Published by Foundation of Computer Science, New York, USA.
16. Amnon Shashua, Yoram Gdalyahu, and Gaby Hayun. Pedestrian detection for driving assistance systems: single-frame classification and system level performance. In *Intelligent Vehicles Symposium, 2004 IEEE*, pages 1–6, June 2004.
17. Jianxin Wu, Wei-Chian Tan, and James M. Rehg. Efficient and effective visual codebook generation using additive kernels. *J. Mach. Learn. Res.*, 999888:3097–3118, November 2011.