Binarization-Based Human Detection for Compact FPGA Implementation

Shuai Xie, Yibin Li*, Zhiping Jia, and Lei Ju

School of Computer Science and Technology, Shandong University, Jinan, China, 250101
xieshuai1210@mail.sdu.edu.cn,
{liyibing,jzp,leiju}@sdu.edu.cn

Abstract. The implementation of human detection in the embedded domain can be a challenging issue. In this paper, a real-time, low-power human detection method with high detection accuracy is implemented on a low-cost field-programmable gate array (FPGA) platform. For the histogram of oriented gradients feature and linear support vector machine classifier, the binarization process is employed instead of normalization, as the original algorithm is unsuitable for compact implementation. Furthermore, pipeline architecture is introduced to accelerate the processing rate. The initial experimental results demonstrate that the proposed implementation achieved 293 fps by using a low-end Xilinx Spartan-3e FPGA. The detection accuracy attained a miss rate of 1.97% and false positive rate of 1%. For further demonstration, a prototype is developed using an OV7670 camera device. With the speed of the camera device, 30 fps can be achieved, which satisfies most real-time applications. Considering the energy restriction of the battery-based system at a speed of 30 fps, the implementation can work with a power consumption of less than 353mW.

Keywords: HOG+SVM, Binarization Process, FPGA Implementation, Low Power Consumption.

1 Introduction

Real-time image-based human detection is an important implementation for vision systems, particularly for embedded environments. Apart from the vision domain, this implementation also has a wide range application prospects in areas such as entertainment, surveillance, robotics, and security. For embedded human-detection applications, real time, detection accuracy, hardware resource requirement, and power consumption are four primary considerations. In many applications, external memory is usually needed. Moreover, a tradeoff must exist between performance and power consumption is trade-off by owing to the limited resources of a field-programmable gate array (FPGA).

* Corresponding author.

C. Wu and A. Cohen (Eds.): APPT 2013, LNCS 8299, pp. 119–131, 2013.

The human detection process primarily contains two significant steps: feature description and classification. During feature description, important information is extracted from the image. The classifier algorithm is used to determine whether a person is present in an image. Various methods for feature description have been proposed, such as Haar wavelets [1], Haar-like features [2], Gabor filers [3], and SHIF descriptors [4]. Likewise, many classifier algorithms are available, such as the support vector machine (SVM) [5] and Adaboost [6]. Nevertheless, these algorithms cannot satisfy the requirements for detection accuracy.

In 2005, the famous histogram of oriented gradients (HOG) feature [7] was proposed, which subsequently became the most widely used algorithm for object detection. This algorithm significantly enhanced the detection accuracy of human detection. However, its high computational complexity has made the HOG algorithm impossible to run on a desktop computer in real time. Numerous hardware implementations of human detection based on HOG algorithm that could work in real time have recently been made. Nevertheless, such methods have always had lower detection accuracy and poor power consumption or required a high-end FPGA for its implementation.

To achieve a good balance of the four considerations mentioned in the first paragraph and to address the concern of having limited resources in embedded implementations such as wireless sensor networks (WSNs), we proposed a simplified human detection algorithm based on the HOG feature and linear support vector machine (SVM) targeting low-end FPGA devices. Binarization is adopted and optimized to replace the normalization process. Additionally, pipeline architecture is introduced to increase the detection speed. Furthermore, few other simplifications and optimizations are introduced during hardware implementation. Finally, our implementation can be mapped on a low-end Xilinx Spartan-3e FPGA and can work in real time, with slightly less detection accuracy and low power consumption.

The remainder of this paper is organized as follows: Section2 reviews related studies on human detection; Section3 provides the architecture of the proposed human detection process; Section4 explains the FPGA implementation details; Section5 recounts the implementation results and evaluation; and Section 6 presents a summary of the work.

2 Related Work

With the extensive literature on human detection, this section mentions only a few relevant papers on the acceleration or hardware implementation of human detection. Our algorithm is primarily based on the original HOG feature algorithm proposed by Dalal et al [7]. However, the original HOG algorithm has a very slow detection rate. In 2006, Zhu et al. [8] proposed a modified human detection algorithm based on a multi-scale HOG feature and a boosted cascade of the Adaboost classifier, which was first proposed in [9]. In this study, the researchers achieved nearly the same detection accuracy as Dalal's implementation that worked in real time, although this algorithm was unsuitable for the hardware used. In 2007, Kerhetet al. [10] proposed a human

detection implementation that had minimal power consumption through the FPGA development board. Although this work was not based on the HOG algorithm, it was implemented well on FPGA with good detection speed and low power consumption. In 2009, Kadotaet al. [11] introduced a hardware implementation of HOG feature extraction. The researchers proposed some ideals of simplification or modification for FPGA implementation and achieved a process speed of 30fps. To reduce the HOG feature size, [12] proposed an effective binarization scheme for the HOG feature. In 2011, Negi et al. [13] employed a deep pipelined architecture for the hardware implementation of human detection. With this architecture, external memory was no longer necessary, and less hardware resources were used. In 2012, Komorkiewicz et al. [14] implemented the original HOG algorithm by using single precision, 32-bit floating point values. Their implementation achieved high detection accuracy, although the use of a high-end Virtex6 FPGA resulted in very high resource utilization.

In the present study, we modified the algorithm used by Negi et al. and further optimized it for hardware implementation, thus achieving significantly improved performance.

3 Human Detection Algorithm

The HOG feature uses the local histograms of oriented gradients of each pixel to characterize the image. This feature expresses the contour of humans and avoids the interference of light and action to a certain extent. The detection process is achieved through the linear SVM classifier. In this study, some modifications were made on these algorithms to suit hardware implementation on FPGA.

The original HOG and SVM algorithms have four steps:

1) Gradient and direction calculation;
2) Histogram generation;
3) Normalization;
4) Classification.

This algorithm is unsuitable for hardware implementation as the dense of square, square root, multiplication, anti-trigonometric, and division operations are calculated during the detection process, and an external memory is always required for the storage of intermediate data. Thus, the binarization process was adopted and optimized to replace the normalization process, resulting in a series of modifications that will be discussed in detail in Sections 3.2 and 3.3. Likewise, other optimizations and simplifications are discussed below.

3.1 Gradient and Direction Calculation

The parameters of the cell and block used for the HOG extraction are 8×8 and 16×16 pixels, respectively. Based on Dalal's work, sample mask (-1,0,1) showed the best performance. Using this mask, the following equation was obtained:

$$\begin{cases} f_x(x,y) = f(x+1,y) - f(x-1,y) \\ f_y(x,y) = f(x,y+1) - f(x,y-1) \end{cases} \tag{1}$$

where $f(x,y)$ represents the luminance value at coordinate (x,y). For enhanced performance, the square root of each channel was obtained, such that each value for f_x or f_y ranges from 0 to 15. The magnitude m and direction θ are calculated by Eqs.(2) and (3). For the color image, the gradients of each color channel were calculated separately, and the largest $m(x,y)$ was considered, as this value is the gradient vector of the pixel.

$$\mathrm{m(x,y)} = \sqrt{f_x(x,y)^2 + f_y(x,y)^2} \tag{2}$$

$$\theta(\mathrm{x,y}) = \tan^{-1}\frac{f_x(x,y)}{f_y(x,y)} \tag{3}$$

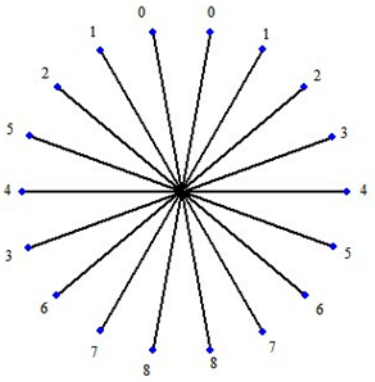

Fig. 1. Quantized gradient directions θ

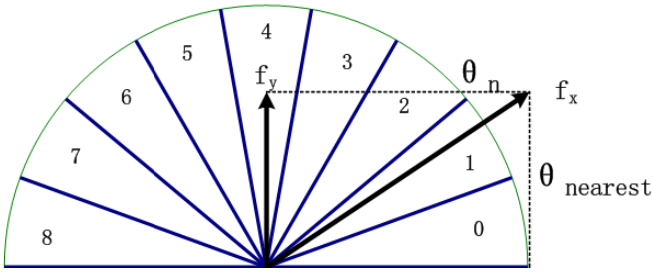

Fig. 2. Nine bins

To compute for the gradient of the orientation histogram, the orientation is divided into nine bins, as shown in Fig.1. For each pixel in a cell, two weighted votes are calculated for the nearest bin and the bin to which the pixel belongs (Fig. 2). The vote is based on the gradient magnitude m, whereas the weight is calculated according to the direction θ. This process is calculated using the following equations:

$$m_n = (1 - \mathrm{a})\mathrm{m}(\mathrm{x}, \mathrm{y}) \tag{4}$$

$$\mathrm{m}_{\mathrm{nearest}} = \mathrm{am}(\mathrm{x}, \mathrm{y}) \tag{5}$$

$$\mathrm{a} = \frac{b\theta(\mathrm{x},\mathrm{y})}{\pi} - (\mathrm{n} + 0.5) \tag{6}$$

In the proposed implementation, a standard RGB-565 image was used. Only the highest four bits from every channel were used for the calculation, which means that only 512 different values are available for gradient m and direction θ. Thus, the weight vote of each pixel can be calculated in advance and then pre-stored in a look-up-table with 1 kb BRAM.

Subsequently, the votes of a cell are grouped and summed according to their direction. Finally, the histogram is generated for each cell.

In this implementation, each block contains four histograms, which is a nine-dimension vector.

3.2 Histogram Normalization

Using steps 1, we obtained serve histograms, each of which is a nine-dimension vector in the hardware, given by:

$$\mathrm{F_{i,j}}=[\mathrm{f^0_{i,j}}, \mathrm{f^1_{i,j}}, \mathrm{f^2_{i,j}}, \mathrm{f^3_{i,j}}, \mathrm{f^4_{i,j}}, \mathrm{f^5_{i,j}}, \mathrm{f^6_{i,j}}, \mathrm{f^7_{i,j}}, \mathrm{f^8_{i,j}}]$$

Each element f_n of $F_{i,j}$ represents the value of bin n in each histogram. This element is called a feature vector. For each cell, we obtained a feature vector, whereas for each block, we obtained a large feature vector consisting of all the feature vectors from the cells that belong to the block. For example, the cell is 8×8 pixels, whereas each block consists of 2×2 cells. Thus, the feature vector of the block is a 36-dimension vector formed by the nine-dimension vectors of the four cells.

To weaken the effect of light and the slight movement of the human body on the feature vector, the feature vector should be normalized. As stated in Dalal's paper, the L2-norm has the best performance, which is given by

$$\mathrm{v} = \frac{V_k}{\sqrt[2]{\|V_k\|^2 + \varepsilon}} \tag{7}$$

where V_k is the feature vector of block k, ε is a constant to avoid division by zero, and v is the final feature vector.

Although this step also has a square root operation, it cannot be realized using a look-up table. The square root operation can be performed using a Cordic IP CORE with a delay of 20 clocks. However, such action would make hardware realization impossible, and a large memory would be necessary to store the feature vector.

In [12], the researchers proposed a binarization process, a method used by [13] with a constant threshold. Although this process degrades performance because of the loss of accuracy of the HOG features, the memory cost is considerably reduced. After normalization, the HOG features of a block become specific values. With this process, the HOG features of each block would have the same weight on the classifier training and detection processes, although the rate of each HOG feature would not be changed

as before. With a constant threshold, the effective features of a block can be highlighted. Moreover, with a threshold that represents the average value of all the features in a block, the same result can be obtained, along with other benefits.

As shown in Fig.3, the red line represents the average value of the 36 HOG features in one block. The final HOG feature is set to 1 if it is greater than the average value and is set to 0 otherwise. This process has two advantages. First, the rate of the HOG features for each block are unchanged with or without the normalization process because this process is no longer required given the selection of an average value as the threshold. Second, the features obtained after the binarization process take the value of either 1 or 0, which will further optimize the detection process with a SVM classifier. This optimization will be discussed in Section 3.3. Additionally, with an average value as the threshold, the same benefit can be obtained as with having a constant threshold.

During the HOG feature generation step, the normalization process costs the most calculation resources given the need to calculate the dense of square, square root, division, and multiplication operations for each block. Assigning average values as the threshold will reduce the resource cost of both the hardware and software.

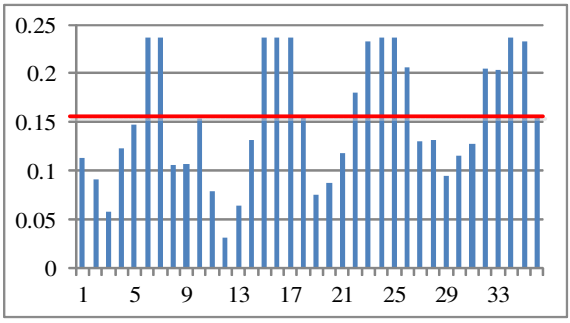

Fig. 3. Binarization process

After the two steps, the HOG feature of the image was obtained.

3.3 SVM Classification

SVM is a machine learning method used for classification and regression analysis. Given a set of training examples, each example is classified under two categories, and an SVM training algorithm builds a model that assigns new examples for each corresponding category.

For the proposed realization, the training data comprised the feature vector of each image, which is a 3780-dimension vector. To simplify the calculation, linear SVM classifier was employed. The SVM classifier was trained offline, and the final SVM classifier was a 3781-dimension vector.

The detection process using a linear SVM classifier involved multiplying the SVM vector by its corresponding HOG features. After the binarization process, the HOG features used to train the SVM classifier took the value of either 1 or 0, and the

multiplication operation was replaced by addition. Statistically, 40% of the HOG features take the value of 1. Finally, in each detection process, 1512addition operations would be calculated, instead of 3780 multiplication and 3780 addition operations. This modification saves hardware resources.

4 FPGA Implementation

An OV7670CMOS video camera was used as the input device. By changing the initial parameters, the input image was fixed at 320×240 pixels, and the frame rate was set as 30fps. Finally, the detection parameters used are shown in Table 1.

Table 1. Parameters

Input image	320x240 pixel
Detection windows	64x128 pixel
Cell	8x8 pixel
Block	2x2 cell
Step stride	8x8 pixel
Number of bins	9

4.1 Gradient Computation

To accelerate the classification process, the pipeline architecture was adopted. As shown in Fig. 4, three lines of three-stage shift registers were used to store four adjacent data during gradient and direction calculation. Two-line BRAM was used to store the other 317 values. As previously mentioned, the calculation of m_n and $m_{nearest}$ was performed using a look-up-table.

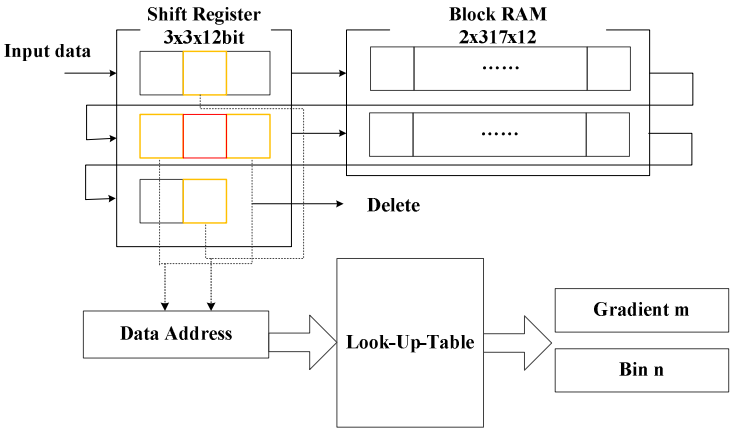

Fig. 4. Hardware structure for the calculation of gradient and bin

4.2 Histogram Generation

For the histogram generation process, pipeline architecture was also used. After the previous process, the weighted votes of each pixel were obtained. The histogram of each cell was generated by summing up the votes of one cell. As illustrated in Fig. 5, a partial histogram was calculated for every eight pixels and then stored in a temporary register. Subsequently, the stream of partial histograms was loaded into the BRAM, such that the partial histograms for eight lines are added up. Thus, the histogram for each cell was generated.

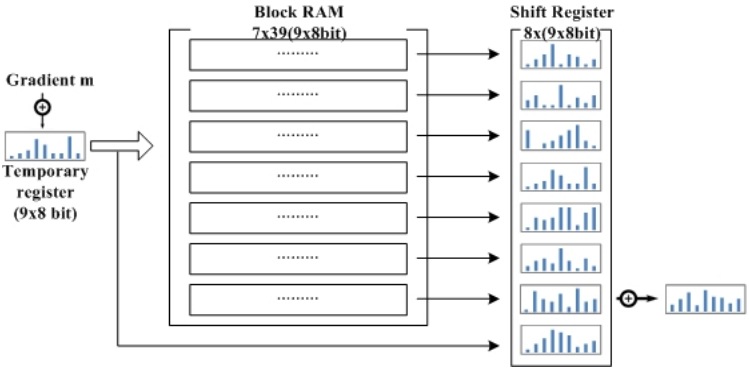

Fig. 5. Hardware structure for the histogram generation

4.3 Histogram Normalization

In our implementation, we adopted optimized binarization instead of the normalization process. The hardware structure and data stream are shown in Fig. 4. This process requires the adjacent feature vectors of four cells. Therefore, two lines of two-stage shift registers and one-line of BRAM buffers were used to store the feature vectors. The average value of the feature vector of each cell was calculated and cached in the temporary register, along with its feature vector. Every time a new feature vector was input, the average value of each block was calculated. Each feature value was then coded in binarization mode.

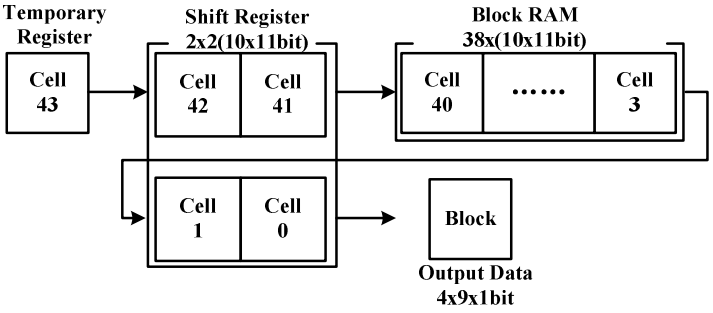

Fig. 6. Hardware structure for binarization

4.4 SVM Classification

With the binarization process, the classification process could be performed by adding the elements of the classifier to a corresponding HOG feature of 1. To accelerate the prediction process, a 3780-dimension filter was built, as shown in Fig.7. The HOG features were stored in the 15×7×(4×9) bit shift registers and the 14×32×(4×9) block ram. The whole HOG feature of a detection window was stored in the shift register and then loaded into the filter. Consequently, the detection results were calculated by adding the SVM elements, which have a corresponding HOG feature of 1. The hardware architecture of this process is shown in Fig.8.

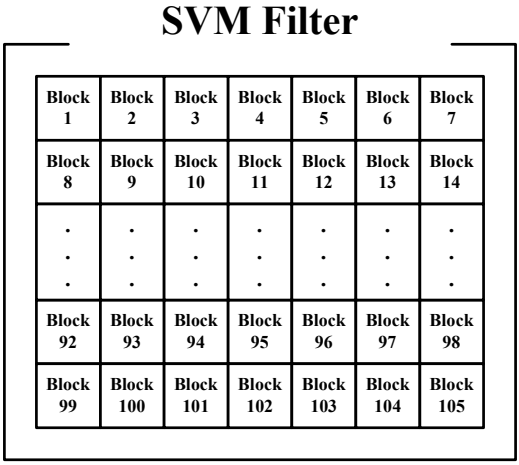

Fig. 7. SVM Filter

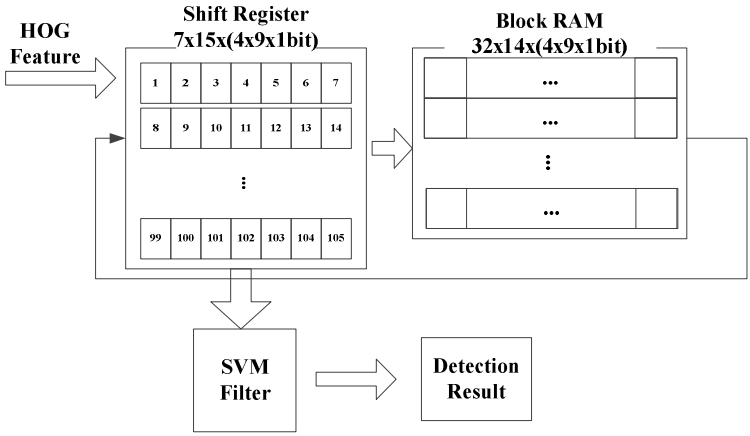

Fig. 8. Hardware structure for the human detection process

5 Implementation Result and Evaluation

The human detection project was implemented in a wireless video development broad made by the authors with a Spartan-3e XC3S500E FPGA (Fig. 9). Anstm32 microcontroller was placed in the board to control the camera and transmit the image data to the FPGA at the correct time. The detection result was transmitted by a 2.4GHz RF24L01B.

Fig. 9. Wireless video development board

Table 2 lists the implementation results of the current study and of Negi et al. The present work showed a reduced resource usage compared with that by Negi et al. Therefore, the present realization could be mapped on this low-end Spartan-3e FPGA with very limited programmable resources.

Table 2. Results oftheFPGA implementation

	Our work on Spartan-3e	Negi's work on Virtex-5	Komorkiewicz's work on Virtex-6
SLICE	2041	2,181	32,428
LUT	3,379	17,383	113,359
FF	2,602	2,070	75,071
BRAM	6	36	119

To capture a stable image data according to the camera device, the present implementation works on 24MHz. At this clock rate, the authors detected 30 input images per second, although this value is far from the limitation of the implementation. In spite of the restricted rate of the camera, our implementation can detect images at293 fps, with maximum frequency of 67.75MHz. The throughput of the FPGA implementation was compared to a software implementation generated by opencv3.1 using a PC with 2.33GHz Intel Core2 E6550 CPU and 4GB DDR2, operated by Windows7. The software implementation achieved about 2.1fps, and

Negi's implementation achieved112fps at a maximum frequency. Therefore, the current implementation is 139.5 times faster than the software implementation, and 2.6 times faster than Negi et al.'s implementation.

Subsequently, the authors re-implemented Negi et al.'s work on software. In this work, eight features are treated as one 8-bit-wide feature during the classifier training process. In contrast, in the current implementation, each HOG feature is used independently. The test results are summarized as a detection error trade-off (DET) curve in Fig. 10.Negi et al.'s work attained a 3.4% miss rate and a 20.7% FFPW. Alternatively, the current implementation attained a1.97% miss rate and 1% FFPW. In comparison with Negi et al.'s work, the detection accuracy of the current implementation was also evaluated by treating every 4 and 2 HOG features as one feature during the classifier training and detecting processes, as shown in Fig.10.Theresults show that this simplification method harmed the detection accuracy. Nevertheless, although the performance of the current implementation is worse than that of the original algorithm, it is still much better than that of Negi et al.

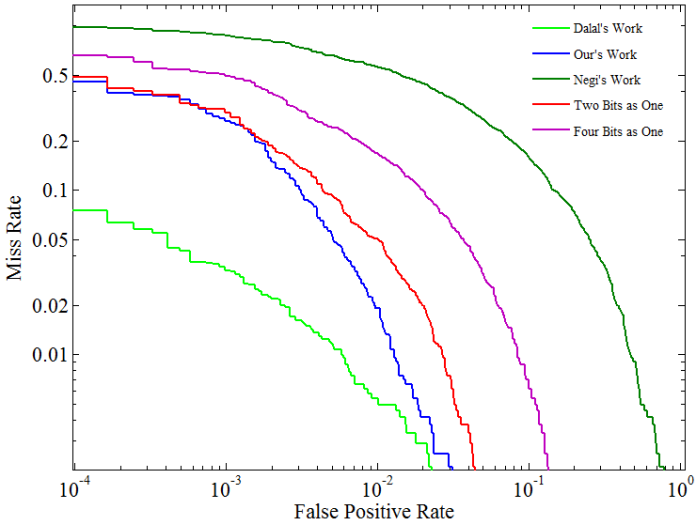

Fig. 10. DET curve for detection accuracy

Table 3. The power evaluation results

Realization	Quiescent Power	Dynamic Power	Total
Ours on Spartan-3e	83mW	270mW	353mW
Ours on Virtex5	444mW	120mW	564mW
Negi's on Virtex5	450mW	438mW	888mW

Finally, we compared the energy consumption of the current implementation and that of Negi et al. The results are summarized using the Xilinx XPower Estimator (Table 3). The current implementation that works on a Spartan-3e has a power

consumption of 353mW, which is extremely low and can satisfy the extreme limitation on a WSN node. Furthermore, using a Virtex5 FPGA, the current implementation achieved 564mW power consumption, whereas that of Negi et al.'s implementation reached 888mW. The quiescent power was almost the same, whereas the dynamic power is nearly a quarter of that of Negi et al.

6 Conclusion

In this paper, a real-time, low power consumption implementation of human detection using the HOG feature and linear SVM was presented. After an experimental implementation on FPGA and an evaluation of the algorithm's detection accuracy through software implementation, the current work achieved a detection rate of 30 fps, with relatively less hardware resources and lower power consumption. Although some simplifications have been made, the detection accuracy is acceptable and relatively higher than that of other implementations. With a high-speed camera, the maximum frequency of 293 fps can be achieved. The current implementation is suitable for the extreme limitation of an embedded platform, such as a WSN node.

References

1. Oren, M., Papageorgiou, C., Sinha, P., Osuna, E., Poggio, T.: Pedestrian detection using wavelet templates. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 193–199 (1997)
2. Viola, P., Jones, M.J., Snow, D.: Detecting pedestrians using patterns of motion and appearance. International Journal of Computer Vision 63(2), 153–161 (2005)
3. Cheng, H., Zheng, N., Qin, J.: Pedestrian detection using sparse gabor filter and support vector machine. In: IEEE Intelligent Vehicles Symposium, pp. 583–587 (2005)
4. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision 60(2), 91–110 (2004)
5. Chang, C.-C., Lin, C.-J.: LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST) 2(3), 27 (2011)
6. Freund, Y., Schapire, R.E.: A desicion-theoretic generalization of on-line learning and an application to boosting. In: The 2nd European Conference on Computational Learning Theory, London, UK, pp. 23–37 (1995)
7. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: The 2005 International Conference on Computer Vision and Pattern Recognition, Washington, DC, USA, vol. 2, pp. 886–893 (2005)
8. Zhu, Q., Yeh, M.-C., Cheng, K.-T., Avidan, S.: Fast human detection using a cascade of histograms of oriented gradients. In: Computer Vision and Pattern Recognition (CVPR), pp. 1491–1498 (2006)
9. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: Computer Vision and Pattern Recognition (CVPR), pp. 511–518 (2001)
10. Kerhet, A., Leonardi, F., Boni, A., Lombardo, P., Magno, M., Benini, L.: Distributed video surveillance using hardware-friendly sparse large margin classifiers. In: Advanced Video and Signal Based Surveillance (AVSS), pp. 87–92 (2007)

11. Kadota, R., Sugano, H., Hiromoto, M., Ochi, H., Miyamoto, R., Nakamura, Y.: Hardware architecture for HOG feature extraction. In: Intelligent Information Hiding and Multimedia Signal Processing, pp. 1330–1333 (2009)
12. Sun, W., Kise, K.: Speeding up the detection of line drawings using a hash table. In: Pattern Recognition (CCPR), pp. 1–5 (2009)
13. Negi, K., Dohi, K., Shibata, Y., Oguri, K.: Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm. In: Field-Programmable Technology (FPT), pp. 1–8 (2011)
14. Komorkiewicz, M., Kluczewski, M., Gorgon, M.: Floating point HOG implementation for real-time multiple object detection. In: Field Programmable Logic and Applications (FPL), pp. 711–714 (2012)