Turkish Journal of Electrical Engineering and Computer Sciences

Volume 24 | Number 5 Article 77

1-1-2016

Hardware implementation of a scale and rotation invariant object detection algorithm on FPGA for real-time applications

MURAT PEKER

HALİS ALTUN

FUAT KARAKAYA

Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2016) 24: 4368 – 4382
© TÜBİTAK
doi:10.3906/elk-1408-187

Hardware implementation of a scale and rotation invariant object detection algorithm on FPGA for real-time applications

Murat PEKER[1,*], **Halis ALTUN**[2], **Fuat KARAKAYA**[1]
[1]Department of Electrical and Electronics Engineering, Niğde University, Niğde, Turkey
[2]Department of Electrical and Electronics Engineering, Faculty of Engineering, KTO Karatay University, Karatay, Konya, Turkey

Received: 26.08.2014 • **Accepted/Published Online:** 07.08.2015 • **Final Version:** 20.06.2016

Abstract: A hardware implementation of a computationally light, scale, and rotation invariant method for shape detection on FPGA is devised. The method is based on histogram of oriented gradients (HOG) and average magnitude difference function (AMDF). AMDF is used as a decision module that measures the similarity/dissimilarity between HOG vectors of an image in order to classify the object. In addition, a simulation environment implemented on MATLAB is developed in order to overcome the time-consuming and tedious process of hardware verification on the FPGA platform. The simulation environment provides specific tools to quickly implement the proposed methods. It is shown that the simulator is able to produce exactly the same results as those obtained from FPGA implementation. The results indicate that the proposed approach leads to a shape detection method that is computationally light, scale, and rotation invariant, and, therefore, suitable for real-time industrial and robotic vision applications.

Key words: Average magnitude difference function, field programmable gate arrays, histogram of oriented gradients, image processing

1. Introduction

Implementation of image processing algorithms on reprogrammable hardware devices with an ability of parallel processing became a 'hot' research topic, and plenty of work has been reported in the literature [1–5]. As the requirement of computationally intensive image-processing algorithms for real-time industrial applications, such as robotic vision, imposes high-speed and highly parallel implementations, in recent years a trend has been witnessed towards utilizing field programmable gate arrays (FPGAs) or graphics processing units (GPUs) instead of using CPUs, which are inherently sequential processing devices. As an indication of this shift, one can encounter various image-processing applications implemented on hardware for real-time applications, such as shape recognition [1,4], medical image-processing [6], object detection [7–10], image retrieval application [11], and image encryption [12]. Parallel computation and reprogrammability are expected to offer a new path to the industrial application of computationally complex applications where shape detection plays a crucial role. Therefore, a new scale-and-rotation invariant shape detection method, which is suitable to implement on FPGA, would be of great interest in terms of its possible applications. In this study, a computationally light but powerful robust method, based on histogram of oriented gradients (HOG) and average magnitude difference function (AMDF), is proposed. Although a similar line of studies is reported in the literature [5,9,13], which

*Correspondence: mpeker@nigde.edu.tr

employ HOG for shape detection, we are proposing a new hardware-friendly algorithm, which employs AMDF as an effective and easy-to-implement decision module on FPGA. The method uses a set of features extracted from the HOG module, which is later classified by the AMDF module as explained in [14–16]. In this paper, we will demonstrate the effective implementation of the proposed method on FPGA for real-time applications. Due to low-power requirements and cost advantages, FPGAs are commonly used in real-time applications. However, FPGA implementation of an algorithm is a very complex task that needs confirmation of the proposed methods in a simulation. Therefore, a MATLAB-based emulation tool [14] will be described, which follows exactly the same design flow and integer arithmetic operations.

The proposed method effectively utilizes the HOG features by tackling the problems related to the hardware implementation of HOG-based shape detection, such as the requirement of nonlinear operations that are difficult to implement on hardware. It is shown in the literature that HOG features demonstrate superior performance in many applications under different and severe conditions [17,18]. However, it is identified that the size of angle bins is a crucial parameter in HOG features. The performance of the proposed method is improved if the size of the bins is increased. The increase in the size of the bins results in more detailed HOG features. However, it is also observed that too much detail deteriorates performance.

The rest of the paper is organized as follows. Section 2 explains the algorithms used in the proposed shape detection. Section 3 explains the hardware simulation environment in MATLAB. Section 4 reports the results obtained in the simulation environment. Section 5 gives the details of the hardware implementation of the proposed method. Section 6 gives the industrial application. Lastly, Section 7 concludes by highlighting the results.

2. Proposed method

The proposed method, which is visualized in Figure 1, incorporates two well-known algorithms, HOG and AMDF, which are frequently used in image and speech processing, respectively. In the method, an input image is initially supplied to the edge detection module. The output of the edge detection module is fed to the module that is responsible for extracting HOG features from the edges of the given input image. An AMDF module is then proposed as an indicator for the differences between the HOG features of the reference images and those of the current input image. The task of the AMDF module is to associate the current input image with one

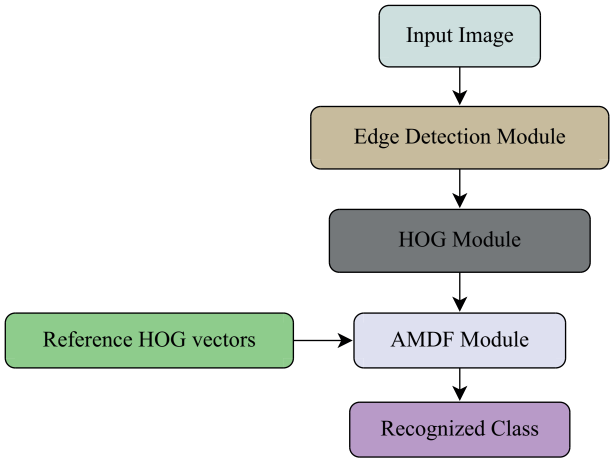


Figure 1. MATLAB flow of the proposed method.

of the reference images based on the similarity of the HOG vectors. Then the current input image is labeled accordingly. In the following sections, the extraction of HOG features and the application of AMDF as a decision module are introduced in detail.

2.1. HOG features

The orientation and magnitude characteristics of pixels can be expressed using HOG features. Therefore, these features could be used to describe an image as a group of local histograms that contain the sum of the gradient magnitudes at the orientation of the gradients in a local region of the image. Our implementation of the HOG algorithm utilized in the proposed method can be given as follows:

i) First of all, edges G_x and G_y are obtained by applying horizontal and vertical Sobel filters, S_h and S_v, as given in Eqs. (1) and (2) on the input image I.

$$G_x = I * S_h \tag{1}$$

$$G_y = I * S_v \tag{2}$$

$$|G_{xy}| = \sqrt{G_x^2 + G_y^2} \tag{3}$$

$$\theta_{xy} = \arctan(\frac{G_x}{G_y}), \tag{4}$$

where the horizontal and vertical Sobel operators are abbreviated as S_h and S_v, respectively, and the amplitude and angle of the gradient at the pixel (x, y) are given as $|G|$ and θ , respectively.

ii) Then the magnitude and orientation angle ($|G_{xy}|$ and θ_{xy}, respectively) are calculated from G_x and G_y using Eqs. (3) and (4).

iii) Lastly, the histogram is constructed by accumulating the gradients at a specific angle to the corresponding bins.

As the size of the bins was expected to be an important parameter that directly affects the performance of the proposed implementation, its effect was investigated. For this purpose, a number of different bin sizes were created between 8 and 360, which gives a rotation angle between 45° and 1°, respectively. In this implementation, HOG features are derived from the whole image, instead of the commonly used implementation of HOG, which divides the image into subregions. Furthermore, the HOG feature vectors are normalized in order to obtain a scale-invariant algorithm by dividing with the maximum HOG value in the current vector, which is formulized in Eq. (5).

$$hog_i = \frac{hog_i}{\max(hog)} \qquad i = 1, 2, ..., N \tag{5}$$

This operation ensures that the maximum value of the angle bins will be set to 1.

2.2. AMDF as a measure of dissimilarity

AMDF is commonly utilized in speech analysis to identify the periodicity of the signal under consideration. Generally, AMDF is used to determine the pitch [19] of a speech in a frame-wise manner. The purpose is to catch the dissimilarity between the speech waveform in a frame and its lagged version. If the waveform has a degree of periodicity, a minimum difference between two waveforms will be obtained at a certain lag that corresponds to the period of the waveform. Therefore, a smaller AMDF value could be interpreted as the similarity between the two waveforms. This feature of the AMDF algorithm might be used to determine the similarity of the rotated waveforms. As the rotation of the image is expected to preserve the general structure of HOG vectors with a shift introduced by the rotation operation equal to the degree of rotation, we might expect that HOG vectors will show circular periodicity under the rotation operation, as seen in Figure 2. Because of this circular periodicity, the approach based on AMDF might be successfully utilized to create a rotation-invariant method, and we propose to use AMDF as a measure of the similarity/dissimilarity between HOG vectors. Furthermore, AMDF requires only addition/subtraction and shift operations; hence its implementation on hardware is expected to be straightforward.

AMDF is defined as a function of the lag value of τ between the *ith* reference HOG and that of *jth* input image in Eq. (6), as follows:

$$f_{AMDF_{ij}}(\tau) = \frac{1}{N} \sum\nolimits_{i=1}^{N} |S_i - S_{j-\tau}| \qquad \tau = 0, 1, ..., N-1, \tag{6}$$

where S_i is the histogram of reference image, $S_{j-\tau}$ the lagged version of the histogram of the input image, and N is the number of bins in the histograms. Please note that the τ value will be in the range of to $N-1$. The reference HOG vectors, associated with a particular class, are extracted and stored in a database. It is possible to determine the class label of the given input object based on the fact that AMDFs will produce a minimum value if the reference object and the rotated input object are similar. Thus, the input object S_j will be classified as an object belonging to the *ith* class, if the function $f_{AMDF_{ij}}(\tau)$ produces the lowest value for the *ith* HOG reference features.

3. Simulation of the method in MATLAB

Hardware implementation of the image and signal processing algorithms is a time-consuming and error-prone process. Thus, instead of implementing the algorithms directly in hardware, it is always preferable to simulate the method beforehand in a high-level language environment such as MATLAB, SystemC, or ImpulseC. In this context, we propose a framework in the MATLAB environment, which provides a flexible platform for experimenting the influence of critical parameters of HOG and AMDF, such as number of bins, bit-width, and number format, on the precision of the results and performance.

Nonlinear computations, such as square root and arctangent operations, are requested to obtain HOG vectors. However, due to hardware constraints, these operations should be implemented using approximations with a proper number format. Decisions on these two parameters (i.e. approximation method and number format) should be based on the consideration of a trade-off between hardware resource utilization, speed, and precision. Therefore, the simulation environment is expected to enable the designer to work with a different number of formats and approximation methods so as to analyze the effect of these parameters on precision and performance. Furthermore, the simulation environment should be able to mimic the hardware environment as much as possible. In order to accomplish this requirement, prior to subsequent operations the format of the input image is converted to 8-bit unsigned gray-scale and the pixel value is set between 0 and 255. The format

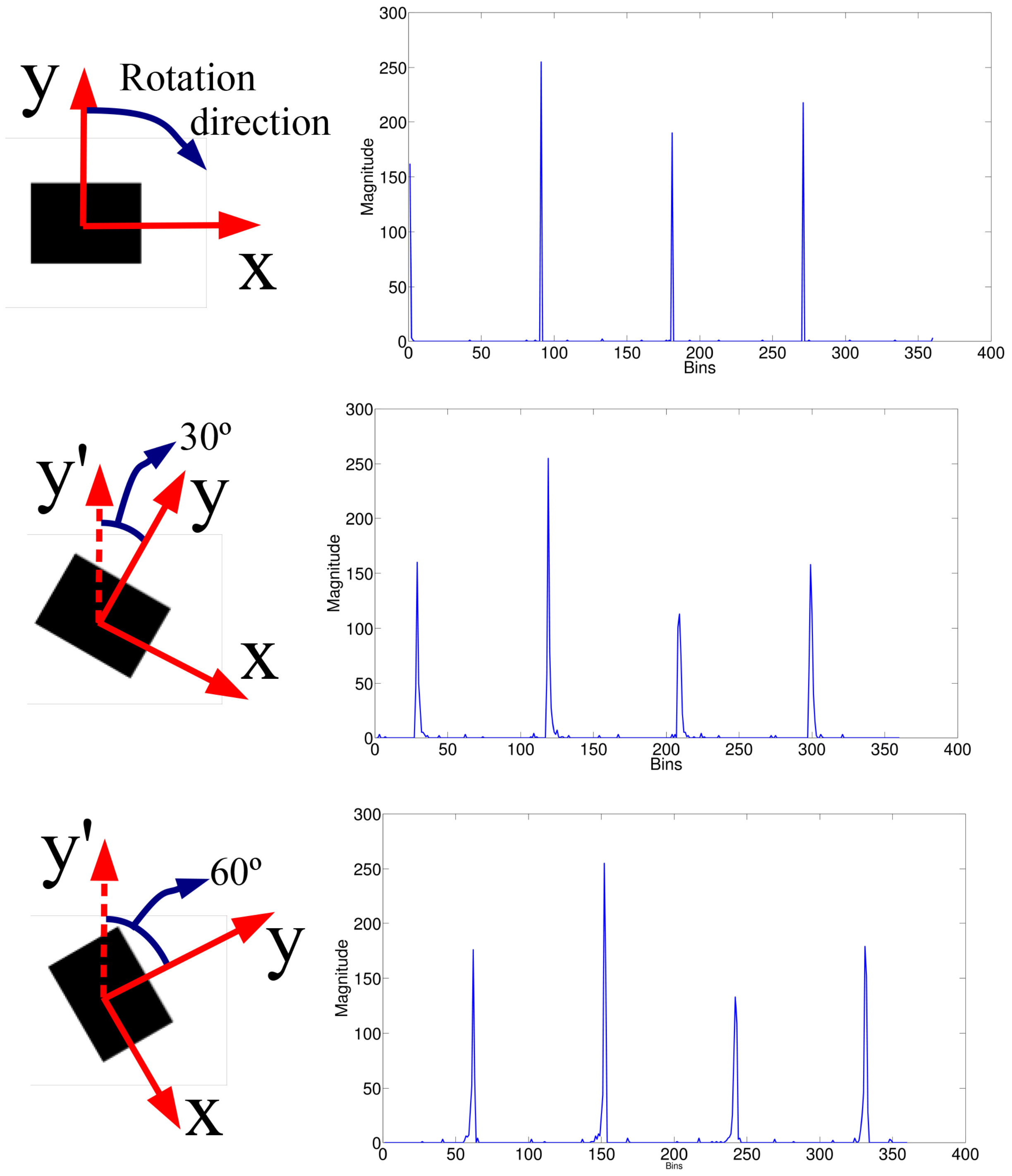


Figure 2. HOG vector for a square shape: **(a)** no rotation, **(b)** clockwise rotation of 30°, **(c)** clockwise rotation of 60°. A circular shift due to rotation is shown clearly.

produced by the Sobel operator is another issue that should be taken into consideration, as these values indicate orientation, which possesses negative numbers. Thus, the outcomes of the Sobel operator are normalized to make sure that the pixel values vary between –255 and 255, represented by 9 digits of signed format.

HOG vectors are obtained using the outcome of the Sobel operations on the input image by feeding these outcomes into two major modules. The angle of the gradient, θ_{xy}, is calculated by the first module using Eq. (3), and the magnitude of the gradient is calculated by the second module using Eq. (4). In order to avoid the nonlinear arctangent operation for the angle calculation, several approximation methods are proposed such as 'piece-wise' [5], 'look up tables' (LUT) [10], and 'CORDIC' [20]. As the accurate calculation of the HOG values mainly depends on the angle of the gradient, θ_{xy}, the angle calculation is highly important for the performance of the proposed method. Considering this requirement, the piece-wise approach, which requires minimal memory, cannot be utilized in approximating the arctangent function, because the error significantly increases for θ_{xy} values greater than 45°. Meanwhile, the LUT approach requires more memory, but the error will be minimal. As a result, for the sake of accuracy, the LUT approach is preferred. In the LUT method, the arctangent values are calculated according to Eq. (7), using the horizontal and vertical gradients (G_x and G_y).

$$\theta_{xy} = arctan(G_x, G_y) = LUT(G_x + G_y * 255) \tag{7}$$

In the proposed method, the calculation of the angle takes place only in the range of 0° to 90°, and, in order to cover the angle values out of this range, the signs of the G_x and G_y are used to determine the region of the angle. Then the angle is mapped according to its corresponding region. In order to improve the performance of the algorithm, a threshold in accumulating the gradient is used, which is set as the mean of the HOG values. If the gradient of the pixel is less than the mean value, this gradient is simply ignored. This restriction is expected to alleviate the problems caused by the contaminated image due to noise and illumination, which is a common problem in the industrial environment.

Another nonlinear operation that imposes high computational load is the square-root operation in the calculation of gradients. In order to avoid this computational load, the square-root operation in Eq. (3) is omitted, and the magnitude of the gradient is given as follows:

$$\mid G_{xy} \mid = G_x^2 + G_y^2 \tag{8}$$

As we obtain HOG features simply by accumulating the gradient values into corresponding bins, omitting the square-root in Eq. (3) is not expected to affect the underlying structure of the HOG features. A further simplification could be the elimination of multipliers and using absolute values instead of taking the square of the values. However, experiments show that using absolute instead of squared values deteriorated the performance because the underlying structure was not well preserved.

The final step is the normalization step, which is defined in Eq. (9).

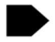
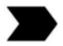
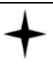
$$HOG_i = (HOG_i \times 255) / \max(HOG) \tag{9}$$

As a result of this normalization, all values in the HOG vector are set in the range of 0 and 255. The results show that the underlying structure of HOG is captured despite the simplified arithmetic operations in the simulation environment. Therefore, due to this well-behaved property, the proposed implementation method is expected to have a high success rate in the detection stage, despite all the simplifications introduced.

4. Simulation results

As the hardware implementation of the proposed method on FPGA is expected to suffer from the number format, which is the bottleneck of the method, it should be addressed carefully. Hence, in this paper, a pure integer number format for the calculations of the Sobel operation and for the extraction of the HOG values is proposed, as explained in Section 3. Before being implemented directly on FPGA, the proposed method was tested on MATLAB, using a dataset constructed with 8 different geometric shapes as reference object classes, which are illustrated along with their corresponding class labels in Table 1. In order to obtain objects with different scale and rotation, all reference shapes were firstly rotated at an angle between 0° and 360° with a step size of 10°. Additionally, the size of each resultant image was upscaled by a factor of 1.5 and downscaled by a factor of 0.75. The size of the images used in the database was 150×110 pixels. The size of the image does not change due to upscaling and downscaling operations, but the size of the shape in the image is scaled accordingly. Since the first class is a circle and rotation for circular shapes is meaningless, the total dataset for class 1 contains only 3 different shapes due to scaling (i.e. original, upscaled, and downscaled). The rest of the classes contain 37 different shapes as a result of rotation. Therefore, the final test dataset size consists of $3 + 37 \times 3 \times 7 = 780$ images with different scales and orientations.

Table 1. Basic shapes with a size of 150×110 pixels in the dataset and their class labels.

Class label	1	2	3	4
Shape				
Class label	5	6	7	8
Shape				

In order to see the effect of the proposed integer operation on the overall performance, and for the purpose of comparison, an exact floating point implementation of the algorithm is also carried out in MATLAB. It is found out that the proposed method in MATLAB simulation has obtained a success rate of 100%. On the other hand, the performance of the hardware implementation is found to be decreased to 96.6% with the same bin size. This degradation in the performance of the proposed method is thought to be a result of the loss of precision due to the number format. It is found out that the majority of the error and degradation of the performance is caused by the fact that the error mostly occurred between certain classes such as classes 6, 7, 3, and 8. Table 2 shows the effect of rotation and the scaling operation on the performance of the proposed method. As seen from the table, the performance of the proposed method for the upscaled version of the images is high. On the other hand, the performance is deteriorated for the downscaled images due to the fact that the underlying HOG structure is not properly obtained in the case of downscaled images.

Table 2. Performance of the algorithm with respect to the degree of rotation and the scale factor.

Angle of rotation	Number of misclassifications			Error rate due to scaling
	No scaling	Downscaling	Upscaling	
0–60	1	4	0	3.87%
60–120	1	2	0	2.32%
120–180	2	5	0	5.42%
180–240	1	2	0	2.32%
240–300	1	3	0	3.1%
300–360	1	3	0	3.1%
Error rate due to rotation	2.69	7.3	0	

The robustness of the proposed methods is tested by introducing a noise in the input images. The various noise levels are created by contaminating 5%, 10%, and 25% of the image pixels with pepper and salt noise, as seen in Figure 3. The results show that the proposed method shows a success rate of 95.95%, 95.89%, and 95.76%, respectively. This high level of success under noise is accounted for by the HOG features, which are noise-immune due to the shape structure of the object, mainly preserved even under a highly noisy environment. As a result, the structure of the obtained histogram still has strong clues, which indicate the class of the object. Hence, the results show that the proposed method is robust against noise, and the performance of the method does not degrade significantly.

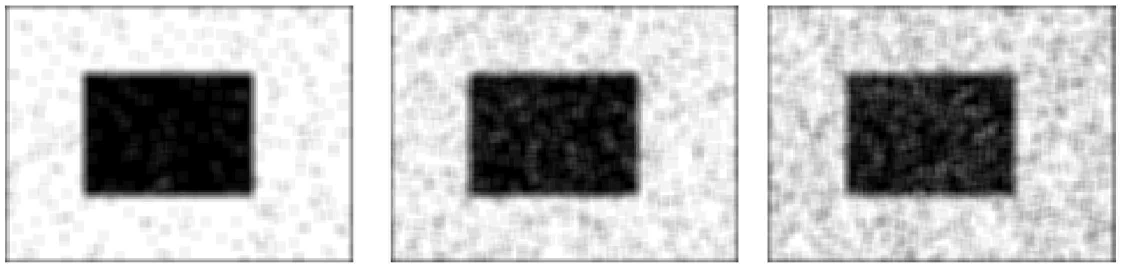


Figure 3. Examples of input images contaminated by pepper and salt noise (5%, 15%, and 25%, respectively).

5. Hardware implementation of the method

After the verification stage of MATLAB, the proposed method is implemented on FPGA using Altera's Cyclone II (EP2C70F896C6N).

A computer is used to transmit the 8-bit grayscale image to FPGA, and the communication interface writes the image on an on-chip memory. Then the Sobel module reads the image from the RAM module and applies Sobel filters. Once the Sobel operation is performed at each pixel, the HOG module calculates both the angle and the magnitude values to generate a histogram. The extracted HOG values are then passed to the AMDF module, which produces a class label at its output as the most similar shape class for a given input image. The detailed block diagram of the design flow and the input/output bit-widths of the modules is illustrated in Figure 4. The input and output bit widths are determined based on the simulation results obtained with MATLAB.

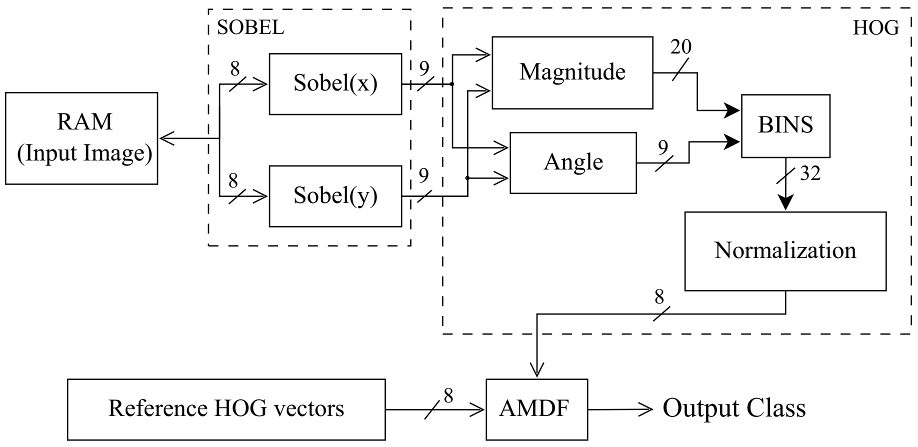


Figure 4. Block diagram of the design flow and bit-widths.

5.1. Implementation of the edge detection module

The Sobel filter in the Sobel module is implemented using 3×3 kernel size. The module performs a prescribed operation on the pixel values, which are read from the internal RAM. P_{11}, P_{12}, P_{13}, P_{21}, P_{23}, P_{31}, P_{32}, P_{33} are the values of the neighboring pixels for P_{22}.

These pixels are 8-bit values written to internal block RAM by the communication module. The Sobel filter module consists of signed substructures and shift registers. In fact, it is possible to use the same module for horizontal and vertical Sobel filtering, but for the sake of performance, two separate units are employed for parallel execution. In the implementation, a 16.5-Kb memory is used to handle the incoming image. An on-chip block RAM structure is utilized for this purpose, and 2 clock cycles are required to read a pixel from the RAM. Applying subtraction and the shifting operations described in Eqs (10) and (11) will produce 11 bits of signed output.

$$G_y = \frac{[(P_{11} - P_{31}) + 2 * (P_{12} - P_{32}) + (P_{13} - P_{33})]}{4} \tag{10}$$

$$G_x = \frac{[(P_{13} - P_{11}) + 2 * (P_{23} - P_{21}) + (P_{33} - P_{31})]}{4} \tag{11}$$

Therefore, to reduce the bit length of the module output, this is divided by 4, resulting in a 9-bit signed output. In hardware implementation, multiplication by two is performed simply by shifting to the left once, and division by 4 is performed simply by shifting to the right twice. Each image of 150×100 requires 15,984 Sobel operations. The Sobel operation for the first pixel of each row takes 21 clock cycles, because 9 pixels must be read from the RAM sequentially (18 clock cycles for reading and 3 clock cycles for arithmetic operations). On the other hand, the rest of the pixels in the corresponding row require only 3 pixels of reading from the RAM, because 6 pixels are already available from the previous Sobel operation. As a result, these Sobel operations require only 9 clock cycles (6 clock cycles for reading and 3 clock cycles for arithmetic operations). The total clock cycles needed for a complete Sobel operation is $108 \times 21 + 15876 \times 9 = 145152$. After the completion of the Sobel operation, the results are directly fed into the HOG module.

5.2. Implementation of the HOG module

The HOG module includes nonlinear operations such as arctangent and square root. For these operations, approximations are employed, which are explained in Section 3. In Figure 5, the top level architecture of the HOG module is illustrated. The inputs of the top module are G_x and G_y, which are calculated by the Sobel module. The DEMUX ($1 \times n$, n is the number of bins) in Figure 5 selects the bin depending on the output of the HOG vector calculator submodule. DEMUX's outputs are the magnitude values, which are added to the corresponding bin cumulatively.

5.2.1. The magnitude calculator submodule

The magnitude calculator is a submodule of the HOG. It calculates the magnitude value of vertical and horizontal Sobel values. Inputs are 9 bits of signed values. The output of the module is 20 bits of unsigned integer.

5.2.2. HOG vector calculator submodule

The HOG vector calculator submodule contains approximations of nonlinear operations, and, therefore, the error rate of this module has a large impact on the performance of the entire algorithm. This module executes predefined conditions, which are given in Table 3 to determine the angle. The module contains a state selector component that outputs the condition number as an input to the MUX, which selects the value as degree.

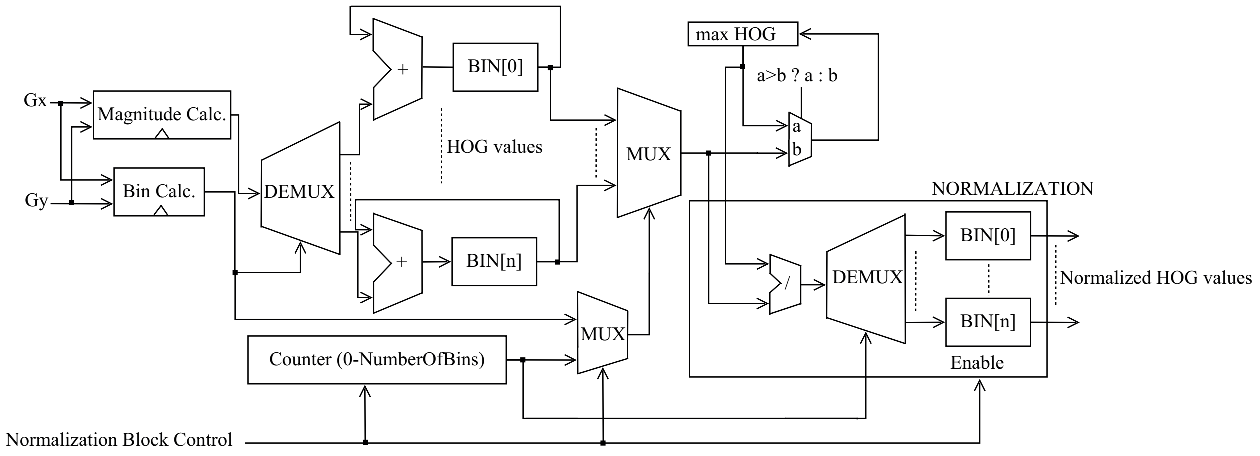


Figure 5. Top module of HOG components.

Table 3. Predefined conditions executed within the HOG vector calculator submodule.

Condition	θ
$\lvert G_x\rvert = \lvert G_y\rvert$	45°
$\lvert G_x\rvert = 0\,\&\,\lvert G_y\rvert \neq 0$	90°
$\lvert G_y\rvert = 0\,\&\,\lvert G_x\rvert \neq 0$	0°
$\lvert G_x\rvert < \lvert G_y\rvert$	$> 45°$ (LUT)
$\lvert G_x\rvert > \lvert G_y\rvert$	$< 45°$ (LUT)

The angle values are read from the LUT, except for the three special conditions $(|G_x| = |G_y|, |G_x| = 0\,\&\,|G_y| \neq 0, |G_y| = 0\,\&\,|G_x| \neq 0)$, and the LUT unit is designed as 73 KB of ROM, in which the address values are constructed by using Eq. (7). After determining the angle, it has to be mapped to its quadrant. For this purpose, a dedicated module called 'map to region' is designed. The module performs the task given in Table 4.

Table 4. Mapping the angle to its corresponding region.

Conditions	Quadrant	θ
$G_x > 0\,\&\,G_y > 0$	I	Deg
$G_x < 0\,\&\,G_y > 0$	II	180 – Deg
$G_x < 0\,\&\,G_y < 0$	III	180 + Deg
$G_x > 0\,\&\,G_y < 0$	IV	360 – Deg

Finally, the angle is divided by variable φ, given in Eq. (12), to get the corresponding bin number.

$$\varphi = 360\,/\,(\text{number of bins}) \tag{12}$$

Based on the operations in the modules explained above, at most 21 clock cycles are needed to calculate the Sobel values of a pixel. Therefore, the HOG and Sobel modules are pipelined with 21-clock cycle latency. After every pixel is placed into its corresponding bin, the normalization stage takes place. Normalization makes the proposed method size-invariant. In addition, because of the strength of the edge magnitude, which depends heavily on the illumination, the normalization process also alleviates the problem arising from the illumination conditions. The normalization module requires 180 clock cycles. Therefore, a total of 145,353 clock cycles are required to complete Sobel, HOG, and normalization. After normalization, bins are stored in registers and are ready to feed into the AMDF module.

5.3. Implementation of the AMDF module

The hardware details of the proposed AMDF implementation are illustrated in Figure 6. In this module, reference HOGs of 8 shape classes are recorded in shift registers, which are capable of rotating the 180 HOG values within 1 clock cycle. To speed up the AMDF operation, the module is constructed to handle 10 HOG values simultaneously. Therefore, the AMDF calculation of the HOG values of a given input image and a reference HOG are completed in 18 clock cycles. Each bin of the input HOG must be subtracted from the bins of the reference HOG one by one, which requires 180×180 (32,400) clock cycles of operation. However, because of the parallel implementation, 10 subtractions are performed simultaneously. As a result, only 180×18 (3240) clock cycles are required. For each class, the calculated partial AMDF values are accumulated to obtain the final AMDF value for the corresponding class. This operation is repeated 8 times to calculate the AMDF values of the stored reference HOG values with respect to the HOG values of the given image. After each calculation, the current AMDF value is checked with the stored minimum AMDF found so far. Meanwhile, the class tag, which corresponds to the current minimum, is stored as an indicator of matching class.

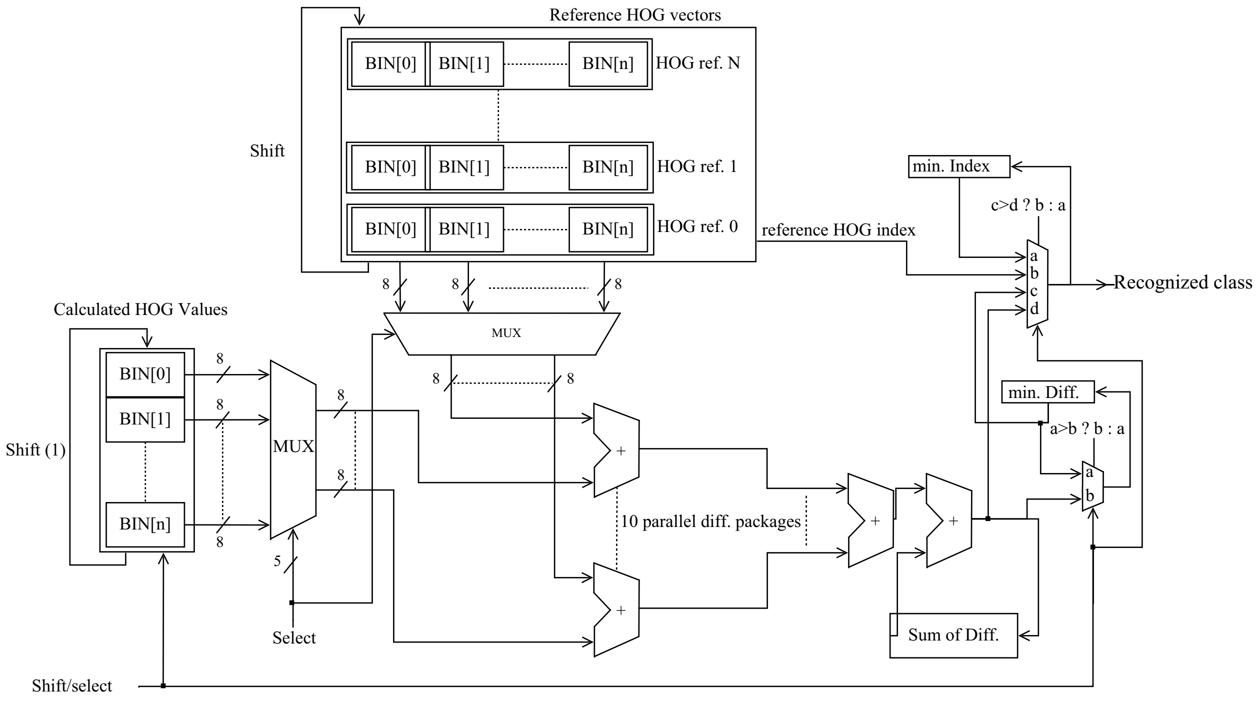


Figure 6. Hardware details of the implemented AMDF module.

The hardware utilization of the proposed method is given in Table 5. The floating point implementation was not possible to fit into the available hardware resources. On the other hand, as seen from the table, the integer implementation of the proposed method consumes only 34% of the available hardware resources with an accuracy loss of 3.4%.

Table 5. Hardware utilization of the proposed method.

Logic utilization	Used	Available	Utilization (%)
Register only	52	68,416	1
Combinational with register	4461	68,416	6
Combinational without register	18,450	68,416	27
Total logic elements	22,963	68,416	34
Total memory bits	82 KB	140 KB	58.5

The details of the module timing are given in Figure 7. As seen from the figure, the Sobel and HOG modules operate almost in parallel, except for some delay of t_1 that corresponds to 21 clock cycles. The total elapsed time, t_2, to finalize the Sobel-HOG module operations corresponds to 145,353 clock cycles. Then, the AMDF module immediately starts processing the available HOG values, which takes only 25,920 clock cycles due to parallel implementation of the module. Total elapsed time to process one test image is given as t_3, which corresponds to 171,273 clock cycles. With a clock of 50 MHz, total elapsed time is calculated as 3.43 ms.

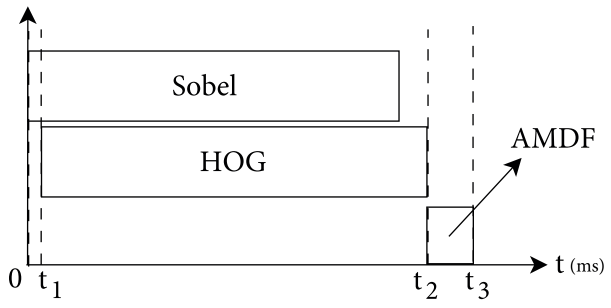


Figure 7. Timing diagram of the modules in the proposed method.

6. Application

The proposed method is tested to recognize geometrical objects on the rotating band, obtained from the images captured by a camera in a harsh industrial environment. The production band has a black and nonhomogeneous background. A camera used for image capturing is placed in an enclosed box to minimize the effect of environmental light changes. The system overview is given in Figure 8.

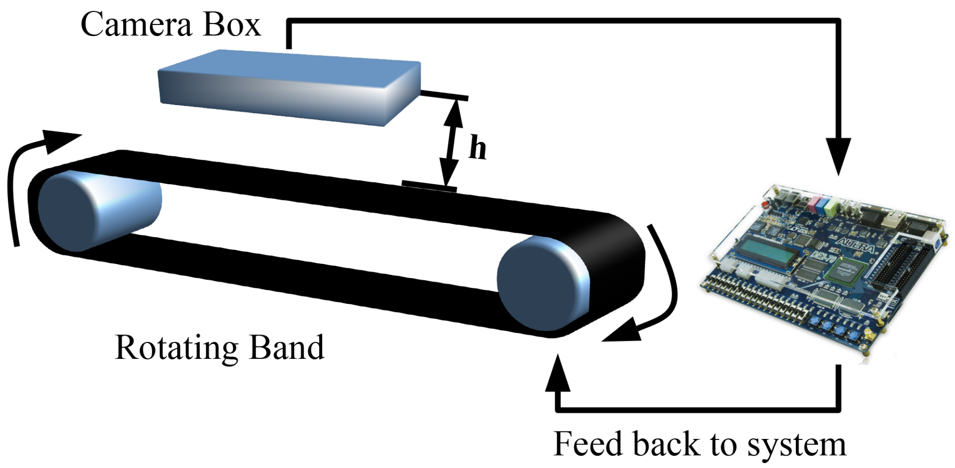


Figure 8. System overview of the production line.

Certain sample images, captured from the production line, are given in Figure 9. The system is used to classify shapes that are made of leather. The system can classify 250 shapes per second. The performance of the system is given in the Conclusion.

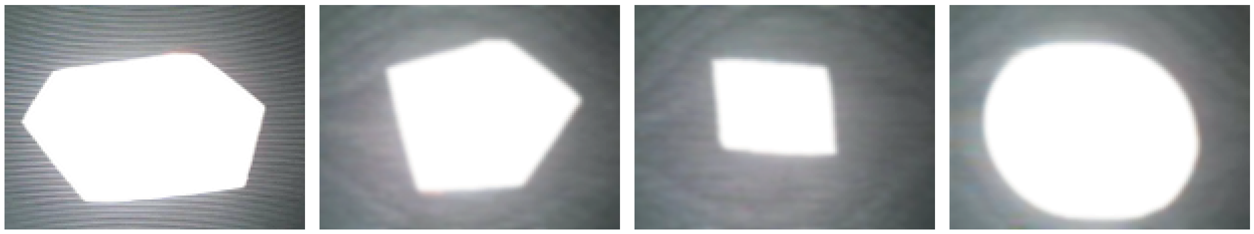


Figure 9. Input images from the production line.

7. Conclusion

The contributions of this paper are three-fold, Firstly, for the first time in the literature a hardware implementation of AMDF is proposed as a measure of similarity/dissimilarity between the HOG vectors of an image in order to determine its class. Secondly, a hardware implementation of a scale and rotation invariant method for shape detection on FPGA is devised. Thirdly, a simulation environment in MATLAB is used in order to overcome the time-consuming and tedious process of hardware verification on an FPGA platform. The simulation environment provides tools to implement the proposed methods, and produces exactly the same results in a simulation environment as in a hardware environment. Initially, the floating point implementation of the method on MATLAB is realized using 180 bins with a success rate of 100%. Then the integer number format implementation of the proposed method is also performed. The number format choice depends on the trade-off between the accuracy and hardware resource. In this study, the integer number format is preferred in order to efficiently utilize the available hardware resources. However, it is also possible to use floating point number format, if there are enough available resources. In the last step, the hardware implementation of the proposed method was performed. It was found that the success rate decreased to 96.6% compared to the floating point implementation, as expected from the simulation. The degradation of performance is thought to be caused by the loss of precision due to the integer number format. Furthermore, to prove the noise immunity of the proposed method, input images are deteriorated by 5%, 10%, and 25% of pepper and salt noise. The performance of the method is found as 95.95%, 95.89%, and 95.76%, respectively, for the given noise levels. The industrial application showed a success rate of 95%, which is very close to the test performance of the method.

It is shown that the performance of the hardware implementation of the proposed method, which utilizes HOG and AMDF as main modules, is highly efficient in terms of speed, accuracy, and hardware resource utilization. It is also shown that the method is noise-immune, size-, and rotation-invariant. Therefore, the proposed method is suitable for real-time robotic vision implementation, which has become a common industrial application due to the availability of hardware resources such as FPGA, which offers highly parallel implementation of the computationally expensive image processing algorithms.

References

[1] Pedrino EC, Morandin O, Kato ERR, Roda VO. Intelligent FPGA based system for shape recognition. In: IEEE 2011 Southern Conference on Programmable Logic; 13–15 April 2011; Córdoba, Spain. New York, NY, USA: IEEE. pp. 197-202.

[2] Chen LD, Zhang MJ, Xiong ZH. Series-parallel pipeline architecture for high-resolution catadioptric panoramic unwrapping. IET Image Process 2010; 4: 403-412.

[3] Geninatti SR, Benítez JIB, Calviño MH, Mata NG, Luna JG. FPGA implementation of the generalized hough transform. In: IEEE 2009 International Conference on Reconfigurable Computing and FPGAs; 9–11 December 2009; Quintana Roo, Mexico. New York, NY, USA: IEEE. pp. 172-177.

[4] Kim D, Jin S, Nguyen DD, Jeon JW. Real-time binary shape matching system based on FPGA. In: IEEE 2008 International Conference on Robotics and Biomimetics; 22–25 February 2009; Bangkok, Thailand. New York, NY, USA: IEEE. pp. 1194-1199.

[5] Karakaya F, Altun H, Cavuslu M. Implementation of HOG algorithm for real time object recognition applications on FPGA based embedded system. In: IEEE 2009 Conference on Signal Processing and Communications Applications; 9–11 April 2009; Antalya, Turkey. New York, NY, USA: IEEE. pp. 508-511.

[6] Koo JJ, Evans AC, Gross WJ. 3-D brain MRI tissue classification on FPGAs. IEEE T Image Process 2009; 18: 2735-2746.

[7] Gao C, Lu S-LL, Suh T, Lim H. Field programmable gate array-based Haar classifier for accelerating face detection algorithm. IEEE T Image Process 2010; 4: 184-194.

[8] Miyoshi T, Shibata T. A hardware-friendly object detection algorithm based on variable-block-size directional-edge histograms. In: IEEE 2010 World Automation Congress; 19–23 September 2010; Kobe, Japan. New York, NY, USA: IEEE. pp. 1-6.

[9] Zhang L, Nevatia R. Efficient scan-window based object detection using GPGPU. In: IEEE 2008 Conference on Computer Vision and Pattern Recognition Workshops; 23–28 June 2008; Anchorage, AK, USA. New York, NY, USA: IEEE. pp. 1-7.

[10] Gu Q, Takaki T, Ishii I. Fast FPGA-based multiobject feature extraction. IEEE T Circuits Syst Vid 2013; 23: 30-45.

[11] Kotoulas L, Andreadis I. Colour histogram content-based image retrieval and hardware implementation. IEEE T Circ Syst 2003; 150: 1350-2409.

[12] Yeo JC, Guo JI. Efficient hierarchical chaotic image encryption algorithm and its VLSI realisation. IEEE T Signal Proces 2000; 147: 167-175.

[13] Kadota R, Sugano H, Hiromoto M, Ochi H, Miyamoto R, Nakamura Y. Hardware architecture for HOG feature extraction. In: IEEE 2009 Conference on Intelligent Information Hiding and Multimedia Signal Processing; 12–14 September 2009; Kyoto, Japan. New York, NY, USA: IEEE. pp. 1330-1333.

[14] Peker M, Altun H, Karakaya F. Hardware emulation of HOG and AMDF based scale and rotation invariant robust shape detection. In: IEEE 2012 International Conference on Engineering and Technology; 10–11 October 2012; Cairo, Egypt. New York, NY, USA: IEEE. pp. 1-5.

[15] Arslan O, Demirci B, Altun H, Tunaboylu NS. A novel rotation-invariant template matching based on HOG and AMDF for industrial laser cutting applications. In: IEEE 2013 International Symposium on Mechatronics and its Applications; 9–11 April 2013; Amman, Jordan. New York, NY, USA: IEEE. pp. 1-5.

[16] Peker M, Altun H, Karakaya F. Hardware emulation of HOG and AMDF based scale and rotation invariant robust shape detection. In: IEEE 2012 International Conference on Engineering and Technology; 10–11 October 2012; Cairo, Egypt. New York, NY, USA: IEEE. pp. 1-5.

[17] Shashua A, Gdalyahu Y, Hayun G. Pedestrian detection for driving assistance systems: single-frame classification and system level performance. In: IEEE 2004 Intelligent Vehicles Symposium; 14–17 June 2004; Parma, Italy. New York, NY, USA: IEEE. pp. 1-6.

[18] Dalal N, Triggs B. Histograms of oriented gradients for human detection. In: IEEE 2005 Computer Society Conference on Computer Vision and Pattern Recognition; 25 June 2005; San Diego, CA, USA. New York, NY, USA: IEEE. pp. 886-893.

[19] Muhammad G. Extended average magnitude difference function based pitch detection. Int Arab J Inf Technol 2011; 8: 197-203.

[20] Andraka R. A survey of CORDIC algorithms for FPGA based computers. In: ACM/SIGDA 1998 International Symposium on Field Programmable Gate Arrays; 22–25 February 1998; Monterey, CA, USA. New York, NY, USA: ACM. pp. 191-200.