PAPER Special Section on Optimization and Learning Algorithms of Small Embedded Devices and Related Software/Hardware Implementation

FPGA Hardware with Target-Reconfigurable Object Detector

Yoshifumi YAZAWA[†], **Tsutomu YOSHIMI**[†], **Teruyasu TSUZUKI**[†], **Tomomi DOHI**[†], ***Nonmembers***,
Yuji YAMAUCHI[††a)], **Takayoshi YAMASHITA**[††], ***and*** **Hironobu FUJIYOSHI**[††], ***Members***

SUMMARY Much effort has been applied to research on object detection by statistical learning methods in recent years, and the results of that work are expected to find use in fields such as ITS and security. Up to now, the research has included optimization of computational algorithms for real-time processing on hardware such as GPU's and FPGAs. Such optimization most often works only with particular parameters, which often forfeits the flexibility that comes with dynamic changing of the target object. We propose a hardware architecture for faster detection and flexible target reconfiguration while maintaining detection accuracy. Tests confirm operation in a practical time when implemented in an FPGA board.

key words: *dynamic reconfiguration of detection target, object detection, FPGA*

1. Introduction

In the field of image recognition, much work is being done on object detection [1]–[3]. Practical object detection technology is in demand in various fields, including ITS, security, and robotics. For computation power required for real-time processing with high accuracy in such applications is beyond the capability of general-purpose computers, so specialized hardware is needed. For that reason, vigorous research is being done on specialized hardware for object detection using graphics processing units (GPU) and field-programmable gate arrays (FPGA) [4]–[9].

Computation time has been an important point in research on hardware implementation of object detection technology. Real-time object detection processing is computation-intensive, and so is difficult to achieve when using hardware that has even lower performance than general-purpose computers.

To overcome that problem, an efficient computational methods and techniques for parallel computation has been proposed [4], [7], [8]. Zhang et alia focused on window overlap when the detection window is moved or scaled during raster scanning and increased the efficiency of feature calculations by re-using the calculation results for the regions of overlap obtained in the first round of calculations [7]. Focusing on the very high computational cost of detection window scanning, Prisacariu et alia achieved high-speed human form detection by parallel computation using a GPU [4]. Kadota et alia achieved faster HOG feature computation by approximation [8].

On the other hand, there has been little work on flexibility, such as the dynamic reconfiguration of the detection target. Object detection [1]–[3] is a technique that targets specific objects, such as humans or vehicles. In doing so, a single algorithm can be used to detect different objects by using different training images and suitable parameters. Past research, however, has taken the approach of optimizing algorithms by using only particular input data and parameters to achieve higher processing speeds. That approach has resulted in a loss of flexibility, such as for dealing with target objects that readily change form. If there are changes in the use environment or the target, the hardware must be redesigned.

In this paper, we propose a hardware architecture that is intended to provide detection target flexibility such as shown in Fig. 1, while maintaining real-time processing performance. With that architecture, the target is easily changed by exchanging previously trained data. For this work, we adopted the joint HOG feature, which is capable of highly accurate object detection [10]. The joint HOG features are calculated and detection is performed on trained data by a classifier constructed of LUTs. It is thus possible to flexibly change the detection target by replacing the content of the LUTs. Furthermore, high processing speed is achieved by using that technique together with pipelining of the calculation stages and parallel processing of windows. Here, we first describe object detection with joint HOG features in Sect. 2 and explain the hardware architecture in Sect. 3. Section 4 evaluates an FPGA board implementation and Sect. 5 presents an example of system operation. Section 6 concludes this paper.

2. Object Detection with Joint HOG Features

Joint HOG features [10] are constructed by using a two-stage boosting process to combine HOG features [2]. This improves discrimination because the HOG features that are effective for detection are selected by boosting. The training process for joint HOG features is illustrated in Fig. 2. This method constructs the final classifier by two-stage boosting. For the boosting, we use Real AdaBoost, in which the weak classifier outputs real values. First, a pool of joint HOG

Manuscript received December 20, 2014.
Manuscript revised May 14, 2015.
Manuscript publicized June 22, 2015.

[†]The authors are with SANEI HYTECHS Co., Ltd., Hamamatsu-shi, 435–0015 Japan.
[††]The authors are with Chubu University, Kasugai-shi, 487–0027 Japan.
a) E-mail: yuu@vision.cs.chubu.ac.jp

DOI: 10.1587/transinf.2014OPP0008

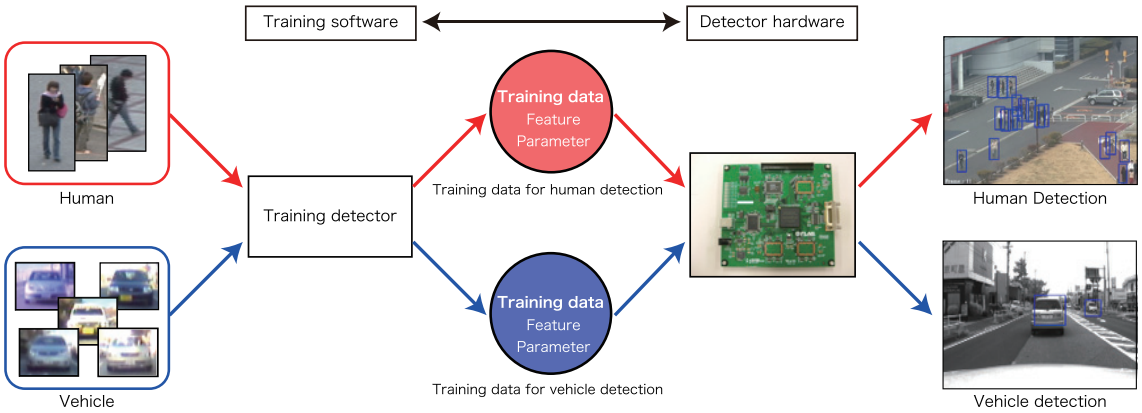


Fig. 1 Hardware with target-reconfigurable object detector.

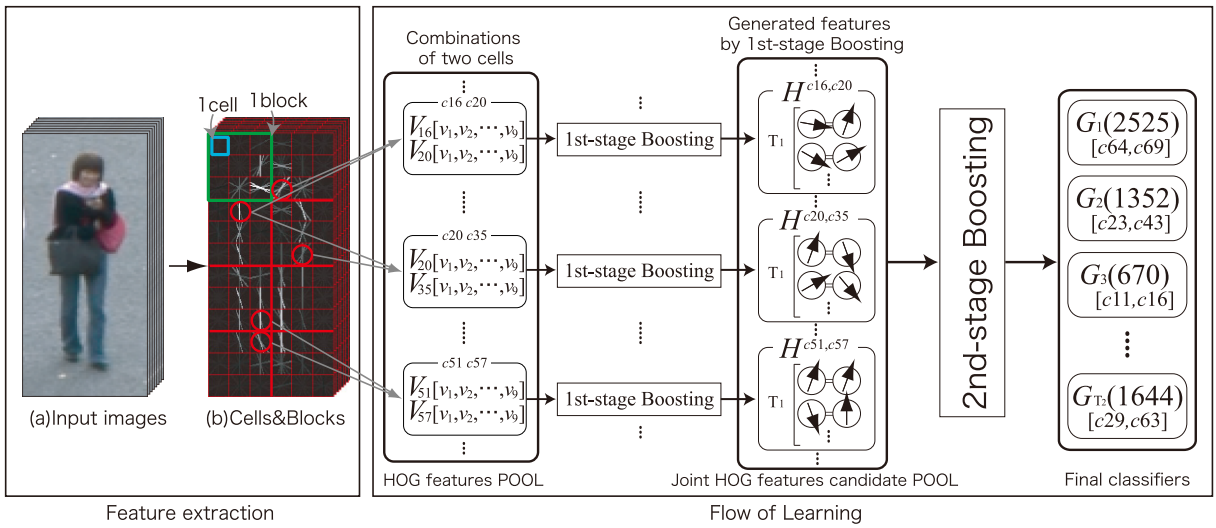


Fig. 2 Overview of training of Joint HOG feature and classifier.

feature, which combine two HOG features of different locations, is created by a one-stage Real AdaBoost with the HOG feature pool as input. Joint HOG futures can represent symmetry and continuous edges because they involve simultaneous observation of multiple HOG features. Next, two-stage Real AdaBoost is used to select joint HOG features from the joint HOG future pool and object detection is performed by the final classifier.

2.1 HOG Feature

In this paper, we use the Histogram of Oriented Gradients (HOG) proposed by Dalal *et al.* as the low-level feature [2]. HOG features are calculated from gradient orientations in local areas called cells (5 × 5 pixels) converted into histograms. They can capture the shape of an object and are robust to changes in illumination and local changes in geometry. The procedure for calculating the HOG features is described below.

From the brightness L of each pixel, compute the gradient magnitude m and orientation θ with the following formula.

$$m(x,y) = \sqrt{f_x(x,y)^2 + f_y(x,y)^2} \tag{1}$$

$$\theta(x,y) = \tan^{-1}\frac{f_y(x,y)}{f_x(x,y)} \tag{2}$$

$$\begin{cases} f_x(x,y) = L(x+1,y) - L(x-1,y) \\ f_y(x,y) = L(x,y+1) - L(x,y-1) \end{cases} \tag{3}$$

The brightness gradient orientation histogram of each cell is generated from the calculated gradient magnitude m and orientation θ. The obtained gradient orientations are divided into 20-degree groups to create the gradient orientation histogram.

Finally, the features are normalized to each block area (3 × 3 cells) with the following equation.

$$v' = \frac{v_j}{\sqrt{\left(\sum_{i=1}^{k} v_i^2\right) + \epsilon}} \qquad (\epsilon = 1) \tag{4}$$

Here, v is the HOG feature, v_j is HOG feature of the j-th dimension of the normalization subject, v' is HOG feature

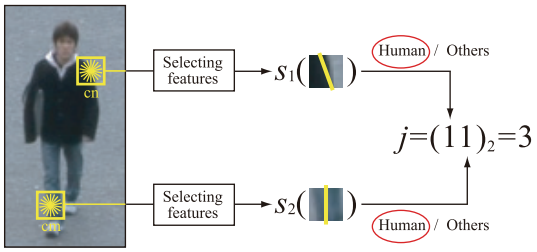


Fig. 3 Co-occurrence of HOG features.

after normalization, k is the number of HOG features in the block, and ϵ is a coefficient for preventing division by zero problems.

2.2 Co-Occurrence Feature

To generate the Joint features, we represent the co-occurrence of multiple HOG features. Representing co-occurrence makes it possible to observe several features(more than two) at the same time. We explain here how to combine two HOG features.

First, we calculate binary symbols s that represent detection objects and non-detection objects with the following equation.

$$s(\mathbf{V}) = \begin{cases} 1 & p \cdot v_o > p \cdot \theta \\ 0 & \texttt{otherwise} \end{cases} \tag{5}$$

Here, θ is the threshold value, p is a parity indicating the direction of the inequality sign, o is the orientation of gradient, and takes the values $p \in \{+1, -1\}$. The value of θ and p are determined so that there error rate is minimized. $\mathbf{V} = [v_1, v_2, \cdots v_q]$ is the HOG feature calculated from one cell, and q is the number of orientation of the gradient in a cell. By combining the two binary symbols obtained in this way, we get features j, which represent co-occurrence. For example, when HOG feature binary symbols $s_1 = 1$ and $s_2 = 1$ are observed in an input image such as shown in Fig. 3, the co-occurrence feature j is $j = (11)_2 = 3$. This j is an index number for a binary representation of combined features. In this case, there are four values because we are dealing with combinations of two features.

2.3 Joint Features

The HOG feature co-occurrence values calculated in Sect. 2.2 are used to generate Joint features in the first-stage Real AdaBoost. This captures the relations of cells as well as the symmetry of object shape and edge continuity.

First, from the features that represent co-occurrence for cells at two different locations, c_m, c_n, Real AdaBoost selects those that are effective in discrimination. Here, a set of N labeled training samples is given as $(x_1, y_1), \ldots, (x_N, y_N)$, where $y_i \in \{+1, -1\}$ is the class label associated with a training sample x_i. The function for observing HOG feature co-occurrence in training sample x_i is expressed as $J_t(x_i)$. Here, t is the number of training rounds. When feature $J_t(x_i) = j$ is observed, the weak classifier $h_t(x)$ of the first-stage Real AdaBoost is expressed as follows:

$$h_t(x) = \frac{1}{2} \ln \frac{P_t(y = +1|j) + \epsilon}{P_t(y = -1|j) + \epsilon}. \tag{6}$$

Here, ϵ is a coefficient for preventing division by zero problems. $P_t(y = +1 \mid j)$ and $P_t(y = -1 \mid j)$ are the respective probability density functions (PDFs) for when the features j that represent HOG feature co-occurrence are observed. The PDFs are calculated with the following equation from the weights $D_t(i)$.

$$P_t(y = +1|j) = \sum_{i:J_t(x_i)=j \wedge y_i=+1} D_t(i) \tag{7}$$

$$P_t(y = -1|j) = \sum_{i:J_t(x_i)=j \wedge y_i=-1} D_t(i) \tag{8}$$

$$D_{t+1}(i) = D_t(i) \exp[-y_i h_t(x_i)] \tag{9}$$

$D_t(i)$ is a weight of a training sample x_i. The weights are initialized by $D_1(i) = 1/N$. The PDFs $P_t(y = +1 \mid j)$ and $P_t(y = -1 \mid j)$ are represented by one-dimensional histograms. The distributions are created by calculating the features that represent co-occurrence from the training samples x and adding the training sample weights D_t to the corresponding one-dimensional histogram BIN[†] numbers j. Because the BIN numbers j correspond to index numbers for the features that represent co-occurrence, there are 4 BIN.

Next, we use the PDF to obtain an evaluation value z_1 that represents the separation of the distributions with the following equation:

$$z_1 = 2 \sum_j \sqrt{P_t(y = +1|j) P_t(y = -1|j)}. \tag{10}$$

Smaller values of z_1 indicate greater separation of the positive class and negative class distributions. The smallest of z_1 is used in the selection of a weak classifier from among the many candidates in each round.

Finally, the Joint feature $H^{c_m,c_n}(x)$, which is the strong classifier of the first-stage Real AdaBoost, is constructed with the following equation:

$$H^{c_m,c_n}(x) = \sum_{t=1}^{T} h_t^{c_m,c_n}(x). \tag{11}$$

The processing described above is applied to all combinations of cells to generate as many Joint features as there are cell combinations. For example, taking a 30×60 pixel input image and a cell size of 5×5 pixels, the number of cell combinations is ${}_{72}C_2 = 2,556$ and 2,556 Joint features $H^{c_m,c_n}(i)$ can be generated. All of the generated Joint features are put into a single pool for input to the second-stage Real AdaBoost to construct the final classifier as described below.

[†]Number of partitions of the histogram.

2.4 Constructing the Final Classifier from the Second-Stage Real AdaBoost

First, we input the Joint features $H^{c_m,c_n}(x)$ and create positive class and negative class probability density distributions W_+ and W_-. The probability density distribution W_+ and W_- is represented by a one-dimensional histogram that is generated from the training sample weights D_t with the following equation:

$$W_+^k = \sum_{i:k\in K\wedge y_i=+1} D_t(i), \tag{12}$$

$$W_-^k = \sum_{i:k\in K\wedge y_i=-1} D_t(i), \tag{13}$$

where, t is the number of training rounds, i is the number of training samples, k is the BIN number of the one-dimensional histogram, and y_i is the class label $y_i \in \{+1, -1\}$. The calculation method of D_t is in the same way as the first-stage Real AdaBoost. The probability density distribution W_+^k and W_-^k can be created by calculating the features from the training samples x_i and applying training sample weights $D_t(i)$ to the BIN numbers k of the one-dimensional histograms that correspond to the feature values. The BIN count of the one-dimensional histogram must be set to a value that is suitable for the number of training samples. In the work reported here, the one-dimensional histogram BIN count was set to 64 by experiment at the second-stage Real AdaBoost. The created probability density distribution W_+^k and W_-^k is normalized so that the sum of all of the probability density distributions of each class is 1.

That probability density distribution is used to obtain the evaluation value z_2, which represents the degree of separation of the distributions.

$$z_2 = 2\sum_j \sqrt{W_+^k W_-^k} \tag{14}$$

Smaller values of z_2 indicate greater separation between the positive and negative class distributions. The minimum value of z_2 is used to select a single weak classifier from among the many candidates in each round.

Next, we use the created probability density distribution W_+^k and W_-^k to calculate the output $g_t(c)$ of the second-stage Real AdaBoost weak classifier. From the values of the HOG feature co-occurrence j obtained from the training samples, we calculate one-dimensional histogram BIN numbers j, and from their corresponding probability density distributions W_+^k and W_-^k we calculate the weak classifier output $g_t(c)$ by using the following equation:

$$g_t(c) = \frac{1}{2}\ln\frac{W_+^k + \epsilon}{W_-^k + \epsilon}. \tag{15}$$

The c is a serial number that represents combinations of cells. The ϵ is a coefficient for preventing division by zero problems and it is given the same value as used in the first-stage Real AdaBoost, $\epsilon = 0.0000001$.

Finally, the following equation is used to construct the final strong classifier $G(c)$ in the second-stage Real AdaBoost.

$$G(c) = \begin{cases} 1 & \sum_{t=1}^{T} g_t(c) > \lambda \\ 0 & \texttt{otherwise} \end{cases} \tag{16}$$

The λ is a threshold value of the classifier. The second-stage Real AdaBoost constructs the classifier by selecting from the Joint feature pool only the features that are effective in discrimination.

3. Hardware Architecture

The hardware architecture in Fig. 4 involves four computational stages from image input to output of the classification result. The computation blocks are described in Table 1. Each block makes use of multiple small-capacity memories to optimize the computation cycle. That greatly reduces the computation wait time. We avoided a reduction in calculation accuracy by using a floating point math library. When designing the hardware architecture, the following objectives shaped the hardware implementation.

- Speed
  - Pipeline processing (optimization of each calculation cycle)
  - Parallel computation of multiple windows
- Flexibility
  - Dynamic reconfiguration of the detection target
  - LUTs of trained data

The hardware implementation is described in detail in the following sections

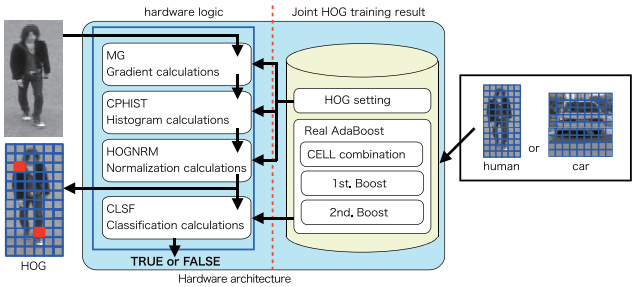


Fig. 4 Hardware architecture.

Table 1 Computation block.

Block	Function	Output
MG	Gradient	Gradient intensity and orientation
CPHIST	Histogram	Histograms of orientated gradients
HOGNRM	Normalization	Normalized histograms
CLSF	Classification	Classification result

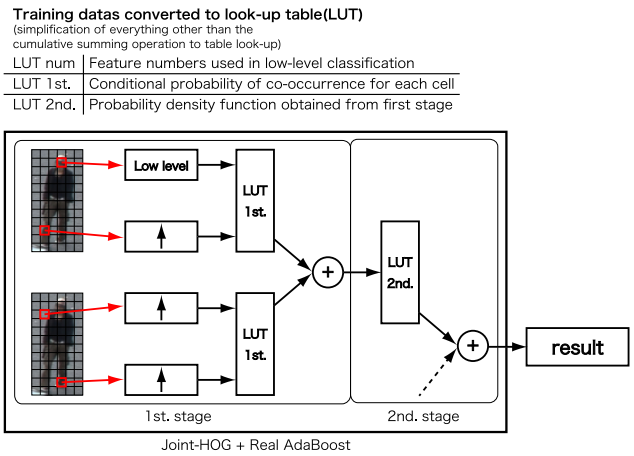


Fig. 5 Object detection with joint HOG features.

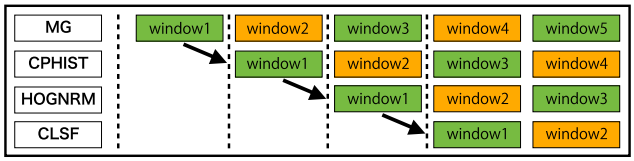


Fig. 6 Pipeline processing.

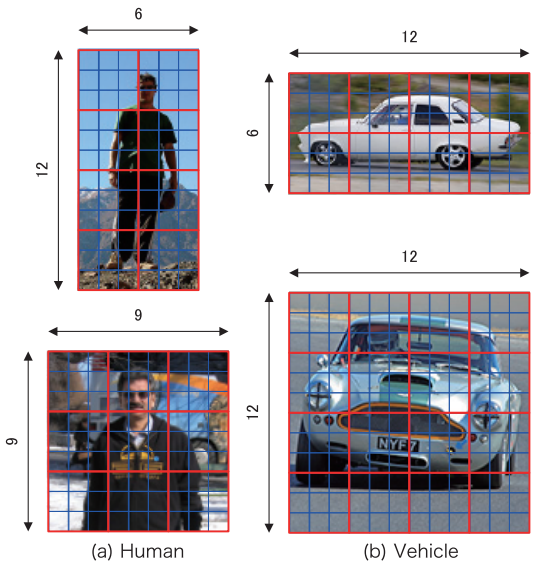


Fig. 7 Examples of multiple models.

3.1 Computing HOG Features

The image data is input and the HOG features are calculated by passing the data through the MG, CPHIST, and HOGRM modules.

In the MG module, a first-order differential is calculated from the image intensity and then a gradient vector (gradient magnitude m and gradient orientation θ) is calculated. To calculate the gradient for one pixel, the four adjacent pixels above, below, and to the left and right of the pixel are used. In our work, we did not handle the data pixel by pixel, but managed the data in quantities of three lines, including the line above and the line below in a FIFO manner for processing in combination with shift registers. By doing so, we were able to calculate the luminance gradients in about as many calculation cycles as for reading the data for one window.

Next, in the CPHIST module, a histogram of orientated gradients is created from the calculated gradient magnitude m, and the gradient orientation θ for each 5×5 pixel cell. That is calculated by cumulative summing for each orientation in an array memory. The number of cells $\times$ orientations are calculated in the number of cycles for reading the amount of data for about one window. Finally, in the HOGNRM module, normalization is performed for each block region. The normalized HOG features are calculated via the squaring, mean square root, and division operations in Eq. (4).

A conceptual diagram of the calculations performed in the CLSF module is shown in Fig. 5. From the HOG features calculated by the HOGNRM module, the combinations of cells that are most suitable for detection are used in Real AdaBoost. The process up to the output of the classification result is shown in Fig. 6. We were able to design the processing performed by the various modules to have about the same computation time. Thus, the process can be pipelined so that calculations can be performed without waiting for the output of subsequent modules.

3.2 Raster Scanning

The proposed method is able to detect a target object from an image using a raster scan. By changing the scale of the window, objects of different sizes can be detected. However, since the computational cost increases when processing multiple scales, we can adjust the scale parameter, if necessary.

Together with the joint HOG classifier, we implemented a raster scanning circuit and a scaling circuit (Fig. 8). First, the input image data that is stored in external memory is copied to internal FPGA memory. That is done to simplify external memory access that involves camera input, detection, and host transfer. The copied images are raster scanned and normalized to provide a constant detection window size. Next, the normalized image is input to the joint HOG classifier.

Furthermore, the joint HOG classifier and the scaling circuit are configured separately. That facilitates parallelization and enables the window size to be changed without affecting computation speed. Normalizing the image for each window also makes it possible to minimize fluctuations in the use of hardware resources.

3.3 Dynamic Target Reconfiguration

We have implemented hardware that allows the detection

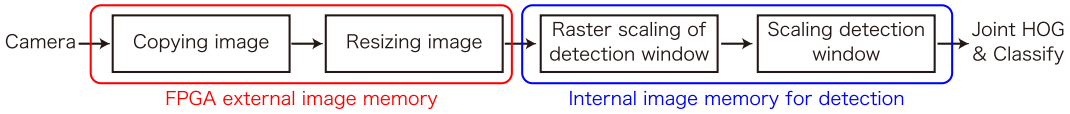


Fig. 8 Object detection process.

object to be easily changed by simply replacing the training data rather than changing the program. In the proposed method, cell size is fixed and the hardware is designed so that the number of blocks within a window can be changed. Changing the number of blocks makes it possible to use a detection window size for detecting features and classification that is suitable even for detection targets that have greatly different aspect ratios, such as human and vehicles.

This approach also makes it possible to apply multiple models for a single target. The appearance of human and vehicles varies greatly according to their orientation (Fig. 7). By loading training data for each of various object orientations into the hardware, it becomes possible to detect a target that is observed from various directions. In addition, the proposed hardware can detect different targets simultaneously.

The target can also be changed easily by exchanging the trained data. In this way, classification can be done with feature combinations that are optimum for the target by exchanging LUTs.

4. Evaluation

The proposed hardware was described in Verilog HDL for logic design. The design was tested by logic simulation and confirmed to produce the same output as a software implementation of the algorithm. Therefore, we confirm that there is no reduction in detection performance due to the hardware implementation. The design was then implemented in an FPGA board and the operation from camera input to detection result output was confirmed.

4.1 Software Processing Time

For comparison of performance, we measured the detection processing time for the hardware implementation and the same joint HOG software implementation. The measurement environment was a Core 2 Duo processor running at 2.33 GHz with 1.95 GB of RAM. The detection parameters and processing times are presented in Table 3.

4.2 Image Processing FPGA Board

The Altera Cyclone III FPGA image processing board that we used is shown in Fig. 9. The detection calculations were performed for the image data input via a camera link. That includes raster scan processing for each detection window in one frame of image data. HOG feature calculations and classification are performed for each window, and both the detection results and the image data are transferred to the personal computer over a USB link. On the personal computer

Table 2 Training parameters.

c_m, c_n	Cell combinations
p	Code for determining inequality sign
θ	Threshold for binaries HOG features
$P_t(y=+1 \mid j), P_t(y=-1 \mid j)$	First-stage PDF
W_+, W_-	Second-stage PDF

Table 3 Parameters and processing time.

Input image	640×480
Normalized detection window size (pixels)	40×80
Detection window (cell)	6×12
Number of scans (pixels)	20
Scaling factor	0.1
Number of scalings	5
Number of windows	2940
Processing time	77.30 ms (about 13 FPS)

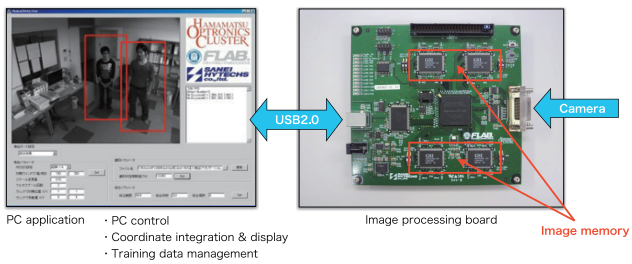


Fig. 9 PC application and image processing FPGA board.

Table 4 Implementation results.

Total number of LE	17,419 (15%)
Total number of registers	11,306 (9%)
Number of internal memory bits	1,046,647bit (26%)
Operating frequency	70MHz
Processing time	93.95 ms (about 10 FPS)
Processing time (two parallel operations)	46.98 ms (about 20 FPS)

side, the received detection coordinates are subjected to window integration processing by mean shift clustering [11] and the window coordinates are displayed. The LUT data shown in Table 2 can also be over-written by the computer via the USB connection.

4.3 Implementation Results

The target results produced by the Cyclone III FPGA (for one detection circuit) and the processing time for one frame (2,940 windows) are shown in Table 4. The software processing time is 77.30 ms, while that of the hardware is 93.95 ms. Thus, the hardware implementation is about 1.2 times

slower. The proposed hardware executes at a high speed because it is capable of pipeline processing. However, the proposed circuit can be implemented as many times as the hardware resources allow. Therefore, higher processing speeds can be achieved by parallel processing. When two parallel processes are used, for example, we confirmed that the detection computation can be done at approximately 20 fps, which is faster than the software processing.

5. Example of Operation

An example training image database is shown in Fig. 10. The organization of the database is shown in Table 5. Negative samples were collected randomly from the background images. We conducted an operation test in which the data obtained by pretraining was read into the internal memory of the FPGA and detection was performed. An example of

Fig. 10 Training image database.

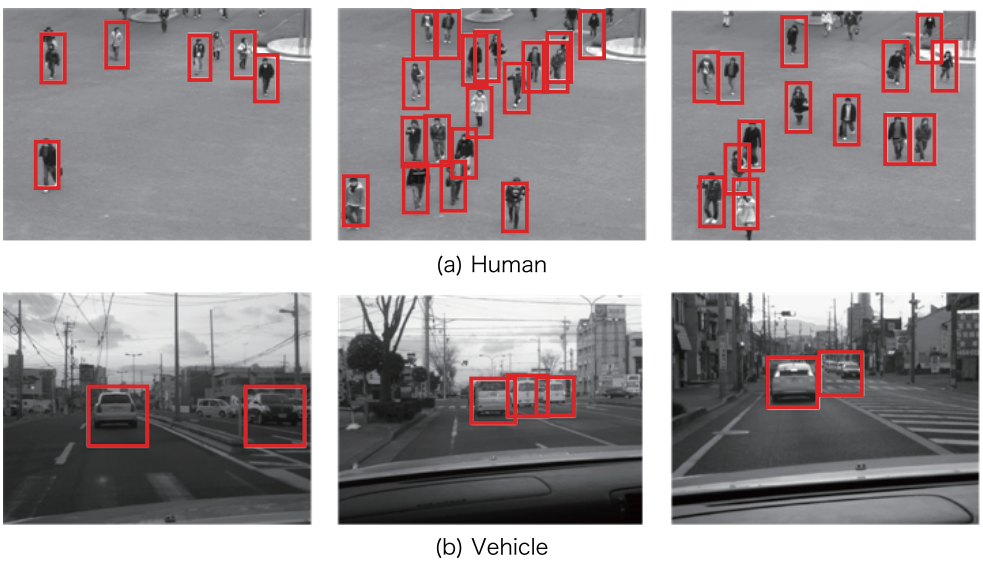


Fig. 11 Examples of detection results using implemented hardware.

Fig. 12 Update detector by retraining.

Table 5 Image database content.

Object class	Positive samples	Negative samples
Human	2,054	6,258
Vehicles	710	8,860

the operation is presented in Fig. 11. The test showed that it is possible to dynamically change the detection target by exchanging the trained data. Because it is possible to transfer the data to the hardware at high speed, the target can be changed easily.

Replacement of the training data is also effective for detector optimization. An example of the operation and experiments on detecting human forms in a different scene is presented in Fig. 12. The training results presented in Table 5 are used for the scene shown in Fig. 12, but there are errors in detection in the part of the image that includes tree branches in the background. It was possible to greatly reduce the detection errors by adding negative samples that include the tree branches and retraining (Fig. 12 (b)).

6. Conclusion

We have proposed here a hardware detector that uses joint HOG features. Easy reconfiguration of the detection target is made possible by exchanging the trained data in the internal LUT of the FPGA. We implemented the proposed system in an FPGA board and confirmed the ability to switch the target between human and vehicles. This detector can perform the computation for one frame of image data in a practical time, but we plan to increase processing speed and reduce size in further development.

References

[1] D.M. Gavrila, "Pedestrian Detection from a Moving Vehicle," European Conference on Computer Vision, pp.37–49, 2000.
[2] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," Computer Vision and Pattern Recognition, pp.886–893, 2005.
[3] P.F. Felzenszwalb, R.B. Girshick, D. McAllester, and D. Ramanan, "Object Detection with Discriminatively Trained Part Based Models," Pattern Analysis and Machine Intelligence, vol.32, no.9, pp.1627–1645, 2009.
[4] V. Prisacariu and I. Reid, "fastHOG - a Real-Time GPU Implementation of HOG," Tech. Rep. 2310/09, Department of Engineering Science, Oxford University, 2009.
[5] B. Bilgic, B.K.P. Horn, and I. Masaki, "Fast Human Detection with Cascaded Ensembles on the GPU," Intelligent Vehicles Symposium'10, pp.325–332, 2010.
[6] K. Mizuno, Y. Terachi, K. Takagi, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "An FPGA Implementation of a HOG-based Object Detection Processor," Information Processing Society of Japan Transactions on System LSI Desin Methodlogy, vol.6, pp.42–51, 2013.
[7] L. Zhang and R. Nevatia, "Efficient Scan-Window Based Object Detection Using GPGPU," Visual Computer Vision on GPU (in conjunction with CVPR), pp.1–7, 2008.
[8] R. Kadota, H. Sugano, M. Hiromoto, H. Ochi, R. Miyamoto, and Y. Nakamura, "Hardware Architecture for HOG Feature Extraction," Proceedings of the 2009 Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing, pp.1330–1333, 2009.
[9] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, and K. Doll, "FPGA-Based Real-Time Pedestrian Detection on High-Resolution Images," IEEE Workshop on Embedded Vision, pp.629–635, 2013.
[10] T. Mitsui and H. Fujiyoshi, "Object Detection by Joint Features based on Two-Stage Boosting," Workshop on Visual Surveillance, pp.1169–1176, 2009.
[11] D. Comaniciu and P. Meer, "Mean Shift: A Robust Approach Toward Feature Space Analysis," Pattern Analysis and Machine Intelligence, vol.24, no.5, pp.603–619, 2002.

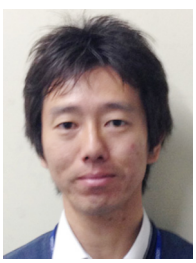
Yoshifumi Yazawa joined SANEI HYTECHS Co., Ltd.

Tsutomu Yoshimi joined SANEI HYTECHS Co., Ltd.

Teruyasu Tsuzuki joined SANEI HYTECHS Co., Ltd.

Tomomi Dohi joined SANEI HYTECHS Co., Ltd.

Yuji Yamauchi received the Ph.D. from Department of Computer Science, Chubu University in 2012. From 2012 to 2014 he was a post-doctoral fellow at the Chubu University. In 2011, he was a visiting student at Robotics Institute, Carnegie Mellon University. He was a Fellowship of the Japan Society for the Promotion of Science from 2010 to 2012. His research interests include computer vision and pattern recognition. He is a member of the IEEE and the IPSJ.



Takayoshi Yamashita received his Ph.D. degree from Department of Computer Science, Chubu University, Japan in 2011. He worked in OMRON Corporation from 2002 to 2014. He is a lecturer of the Department of Computer Science, Chubu University, Japan since 2014. He current research interests include object detection, object tracking, human activity understanding, pattern recognition and machine learning. He is a member of the IEEE and the IPSJ.

Hironobu Fujiyoshi received his Ph.D. in Electrical Engineering from Chubu University, Japan, in 1997. From 1997 to 2000 he was a post-doctoral fellow at the Robotics Institute of Carnegie Mellon University, Pittsburgh, PA, USA, working on the DARPA Video Surveillance and Monitoring (VSAM) effort and the humanoid vision project for the HONDA Humanoid Robot. He is now a professor of the Department of Computer Science, Chubu University, Japan. From 2005 to 2006, he was a visiting researcher at Robotics Institute, Carnegie Mellon University. His research interests include computer vision, video understanding and pattern recognition. He is a member of the IEEE and the IPSJ.