Article Information

Title	FPGA Implementation of Exclusive Block Matching for Robust Moving Object Extraction and Tracking
Authors	Yoichi TOMIOKA, Ryota TAKASU, Takashi AOKI, Eiichi HOSOYA, and Hitoshi KITAZAWA
Citation	IEICE TRANSACTIONS on Information and Systems, Vol.E97-D, No.3, pp.573-582
Copyright	copyright@2014 IEICE
IEICE Transactions Online URL	https://search.ieice.org/

PAPER

FPGA Implementation of Exclusive Block Matching for Robust Moving Object Extraction and Tracking

Yoichi TOMIOKA[†a)], ***Member***, **Ryota TAKASU**[†b)], ***Nonmember***, **Takashi AOKI**[††c)], **Eiichi HOSOYA**[††d)], ***and*** **Hitoshi KITAZAWA**[†e)], ***Members***

SUMMARY Hardware acceleration is an essential technique for extracting and tracking moving objects in real time. It is desirable to design tracking algorithms such that they are applicable for parallel computations on hardware. Exclusive block matching methods are designed for hardware implementation, and they can realize detailed motion extraction as well as robust moving object tracking. In this study, we develop tracking hardware based on an exclusive block matching method on FPGA. This tracking hardware is based on a two-dimensional systolic array architecture, and can realize robust moving object extraction and tracking at more than 100 fps for QVGA images using the high parallelism of an exclusive block matching method, synchronous shift data transfer, and special circuits to accelerate searching the exclusive correspondence of blocks.

key words: *object tracking, motion extraction, FPGA, synchronous shift data transfer, linear assignment*

1. Introduction

Moving object extraction and tracking are essential techniques for image sequence analyses, such as motion recognition and key frame extraction. Since a large amount of computation is required by tracking algorithms robust to illumination change and appearance change due to movement, parallel computations using special purpose hardware are necessary for real-time tracking. However, the category of algorithms accelerated by hardware implementation is constrained; the algorithm should be suitable for parallel processing, the size of computing units should be small, memory contention should be avoided, and so on. In this paper, we present an implementation of exclusive block matching that realizes robust moving object extraction and tracking, which is essentially an algorithm designed for hardware implementation.

Various hardware implementations for extracting features such as SIFT and HOG have been proposed [1]–[3]. In order to realize robust moving object tracking, it is important to obtain the appropriate correspondence of the extracted features. However, these hardware implementations do not handle feature matching. In recent years, due to the growing importance of tracking hardware, various tracking hardware implementations have been proposed. Tracking algorithms can be classified into the following three categories according to the resolution of the extracted motions: (a) pixel-level tracking methods, such as optical flow [4]–[7]; (b) tracking methods for obtaining the correspondence between small areas, such as block matching [8]; and (c) object-level tracking methods, such as mean-shift [9] and particle filter [10], [11]. In previous studies [12], [13], the authors developed optical flow hardware that belongs to category (a); this hardware can extract detailed motion but cannot handle large displacements of moving objects. Moreover, this hardware can realize only existing and basic functions, and does not show the experimental results of benchmark data in a real environment. In Ref. [14], hardware implementation of a mean shift-based object tracking method that belongs to category (c) was presented. This method can handle large displacements of moving objects, but cannot obtain the motion of each part of the objects. For analyzing human action and industrial robot behavior, the tracking methods in category (b) are effective since they can realize detailed motion extraction as well as moving object tracking with large displacements.

The authors proposed the exclusive block matching method (EBM) [8], which realizes the function of category (b). The EBM method [8] is based on the exclusive correspondence between blocks of identical size, and has the following characteristics. (1) The correspondence between each part of a moving object can be obtained, as well as the optical flow, as shown in Fig. 1. (2) Even moving ob-

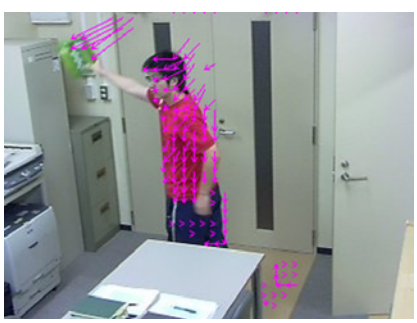


Fig. 1 Example of human motions extracted by exclusive block matching method [8].

Manuscript received April 24, 2013.
Manuscript revised October 18, 2013.

[†]The authors are with the Department of Electrical and Electronic Engineering, Tokyo University of Agriculture and Technology, Koganei-shi, 184–8588 Japan.
[††]The authors are with NTT Microsystem Integration Laboratories, NTT Corporation, Atsugi-shi, 243–0198 Japan.
a) E-mail: ytomioka@cc.tuat.ac.jp
b) E-mail: 50011257509@st.tuat.ac.jp
c) E-mail: aoki.takashi@lab.ntt.co.jp
d) E-mail: hosoya.eiichi@lab.ntt.co.jp
e) E-mail: kitazawa@cc.tuat.ac.jp
DOI: 10.1587/transinf.E97.D.573

jects with large displacements can be tracked, as in a particle filter. (3) Robust moving object extraction and tracking can be achieved using the exclusive correspondence between blocks and considering the relative block positions in an identical object. (4) Most processing, such as similarity calculation, can be performed in parallel, and thus this method can be effectively accelerated by hardware implementation. Although the extracted flows are quantized at the block level in the EBM method, in our experiments, we show that the extracted flow can be applicable for human motion analysis. Furthermore, if necessary, the resolution of the extracted flow can be increased using overlapped blocks.

When the EBM method is executed on 3.14 GHz CPU, the frame rate is about 3 fps for QVGA images. For real-time tracking, a hardware acceleration of the EBM method is indispensable. In order to realize high-speed processing of the EBM method on hardware, we develop a two-dimensional SIMD array processor based on a systolic array architecture that has the following functions: (1) Synchronous shift data transfer for inter-block data transfer; (2) savings-regret approximation [15] and winner-take-all circuit [16] for linear assignment; and (3) parallel connected component detection for object ID estimation. We implemented on FPGA this hardware, which achieved robust moving object extraction and tracking at more than 100 fps for QVGA images by utilizing the above functions. If we implement this hardware on ASIC, we may get a better performance. However, ASIC is much more expensive than FPGA unless producing in extremely large volume. Thus, in this paper, we selected FPGA implementation.

In the next section, we describe the outline of exclusive block matching. In Sect. 3, we explain the implementation of the proposed tracking hardware. In Sect. 4, we show the results of the hardware implementation, and demonstrate the performance of the tracking hardware. Finally, we present our conclusions in Sect. 5.

2. Exclusive Block Matching

2.1 Outline of Exclusive Block Matching

The exclusive block matching method proposed in Ref. [8] divides each frame into blocks of 8×8 pixels, and the detailed motion of moving objects is estimated at the block level. We calculate the similarities between the blocks in the current frame and those in the background and previous frame. This problem is formulated as the following linear assignment problem with the cost matrix shown in Fig. 2.

minimize

$$\sum_{i,j} C(i,j)x_{ij}, \tag{1}$$

subject to

$$\sum_{j} x_{ij} = 1 \qquad \forall i, \tag{2}$$

$$\sum_{i} x_{ij} \le 1 \qquad \forall j, \tag{3}$$

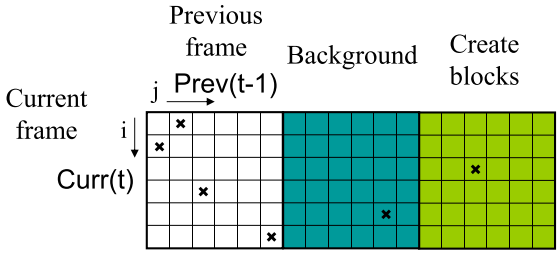


Fig. 2 Example of cost matrix in the exclusive block matching method. The check marks indicate that $x_{i,j} = 1$ in Eqs. (1)–(4).

$$x_{ij} \in \{0, 1\} \qquad \forall i, j, \tag{4}$$

where x_{ij} denotes whether the i-th block in the current frame is assigned to the j-th block in the previous frame, background, or newly created block, and $C(i, j)$ denotes the similarity between them.

The cost matrix shown in Fig. 2 can be divided into three parts. The first part includes the similarities between the blocks of the current and previous frame. The second part includes the similarities between the blocks of the current frame and background. The third part includes threshold values for regarding a block as part of a new object. The blocks in each frame are sorted in the raster scan order in the cost matrix. In the second and third parts, only diagonal elements can be selected.

2.2 Exclusive Block Correspondence

Under the assumption that the shape of an object does not change abruptly and an object does not scale up/down abruptly, the blocks in the current frame almost correspond one-to-one with the blocks in the previous frame. In the exclusive block matching method, each block of the current frame is exclusively assigned to a block of the previous frame, background, or created block. Even if the features of multiple blocks are similar, exclusive block matching allows them to avoid matching with a single block and getting too close.

2.3 Block Features for Similarity Calculations

Many features exist: HSV, HOG, SIFT, SURF, and so on. Though the hardware for generating HSV and HOG features is smaller than that for generating dense SIFT features, the EBM method with HSV and HOG features can achieve as high performance as that with dense SIFT features. It will be demonstrated in Sect. 4.2. In order to save hardware resource, we select HSV and HOG for our hardware implementation.

HSV and HOG are combined for measuring the similarity between blocks. In the original EBM method [8], the Bhattacharyya distance is employed; however, the sum of the absolute difference was employed in this study since the Bhattacharyya distance requires square root calculation and it wastes hardware resource. We will evaluate the effects of this simplification in Sect. 4.2. The combined cost C_{Block}

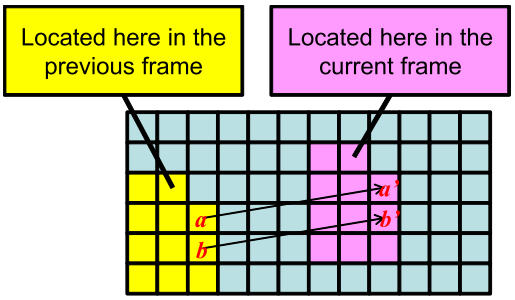


Fig. 3 Cost modification by adding the similarities of eight neighbor blocks.

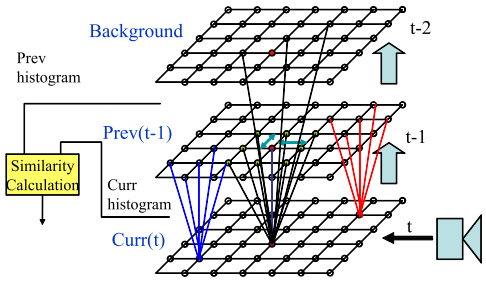


Fig. 4 Similarity calculation for moving object extraction and tracking through consecutive frames.

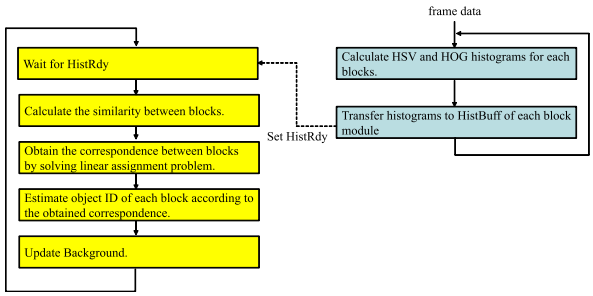


Fig. 5 Processing flow of exclusive block matching hardware.

was calculated as

$$C_{\mathrm{Block}} = \alpha C_{\mathrm{HSV}} + (1-\alpha) C_{\mathrm{HOG}} \tag{5}$$

where C_{HSV} and C_{HOG} are the similarities of HSV and HOG features, respectively, and α is a parameter to balance them which was set as 0.5 in our implementation.

2.4 Consideration of the Similarities of Eight Neighboring Blocks

After the combined cost C_{Block} is calculated for each candidate of block correspondence, it is modified as follows. In Fig. 3, when block a in the current frame corresponds to block a' in the previous frame, block b is also similar to block b'. Therefore, we consider the similarities between the eight neighboring blocks of a if they belong to the same object as a. The matching cost, C_8, of a and a' is calculated as

$$C_8(a,a') = (1-\lambda) C_{\mathrm{Block}}(a,a') + \lambda \frac{1}{|\mathbf{N}_a|} \sum_{b \in \mathbf{N}_a} C_{\mathrm{Block}}(b,b'). \tag{6}$$

Here, $\mathbf{N}_a$ is the set of blocks which are the eight neighboring blocks of a and are in the same object as a. The relative position of b' from a' is the same as the relative position of b from a. λ is the weighting factor and was set as 0.5 in our implementation.

3. Hardware Implementation

3.1 Outline of Exclusive Block Matching Hardware

As explained in Sect. 2.1, for moving object tracking, each frame is divided into blocks of 8×8 pixels, and the HSV and HOG histograms are calculated for each block. The similarities between each current frame's block and its neighboring previous frame's blocks are calculated as shown in Fig. 4. Each previous frame's block is deemed to move to the position of a similar current frame's block by solving linear assignment problem.

The similarity calculation for each current frame's block can be performed in parallel. The processing of two-dimensional PE array is suitable for such parallel computation; the processing for a block of i-th row and j-th column in the images is assigned to the PE of i-th row and j-th column. Three HSV and HOG histograms of a block are stored in a memory of the corresponding PE for the current frame, the previous frame, and the background. For similarity calculations, it is required for each PE to receive the HSV and HOG histograms from its neighboring PE. That is bottleneck to realize highly parallel computations on the two dimensional PE array. In order to solve this bottleneck, we propose *synchronous shift data transfer* technique on the two dimensional PE array, which is a kind of systolic array processor, with a torus network. Because of a torus network, the histograms of all PEs can simultaneously be transferred to their adjacent PEs without losing any histograms even on the boundary of the PE array. By iterating simultaneous and identical transfer, all PEs can receive the histogram of one of their neighboring PEs from their adjacent PE at the same time. The details of synchronous shift data transfer technique will be explained in Sect. 3.2.

The processing flow of exclusive block matching is shown in Fig. 5, and the block diagram of our system is shown in Fig. 6. YCbCr data received from a camera module are converted to RGB data, and stored in a line buffer of the *Histogram Controller*. The RGB data and intensities are sent to *HSVGen* and *HOGGen*, respectively. The level of H, S, and V are set to 12, 8, and 12, respectively, and the HSV histogram of a block consists of 32 (12 + 8 + 12) bins. The HOG histogram of a block also consists of 32 bins from four sub-areas consisting of 8 bins. The bit width of each bin is 8. The system has multiple pairs of *HSVGen* and *HOGGen*, each of which simultaneously calculates the HSV and HOG histograms of blocks in the assigned columns. The two-dimensional PE array, which is a kind of systolic array processor, is based on SIMD architecture, and has ad-

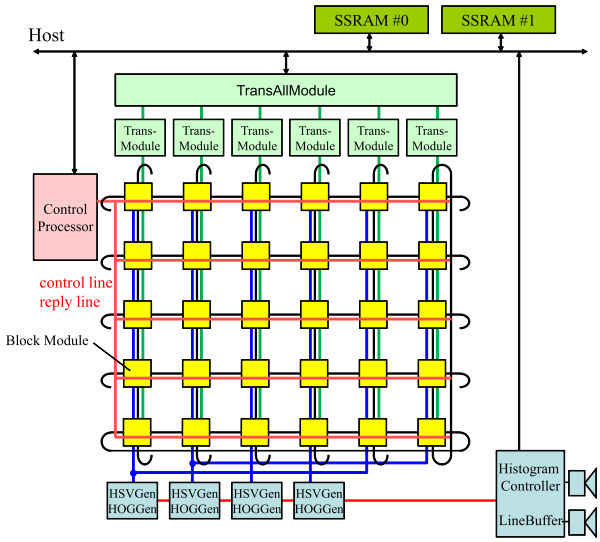


Fig. 6 Block diagram of exclusive block matching hardware.

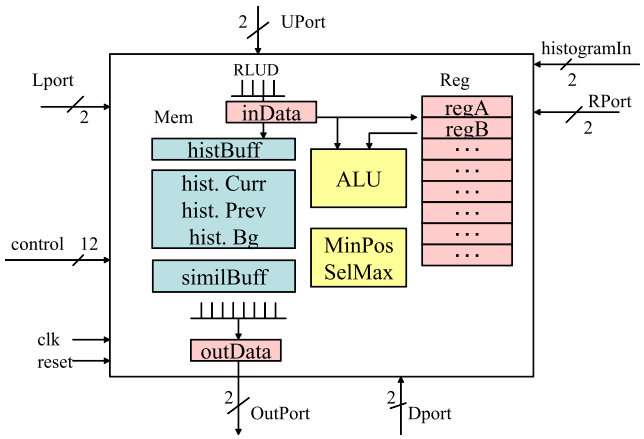


Fig. 7 Block diagram of *Block Module*.

jacent PE-PE connections based on a torus network. Each PE corresponds to a block of 8×8 pixels in the images, and it is called a *Block Module*. The calculated histograms of a block are stored in the internal memory of the corresponding *Block Module*. The *Block Modules* are controlled by a *Control Processor*. Block modules can transfer data to an external SSRAM via *TransferModules* in parallel, and the results of tracking are sent to a host PC via the external SSRAM, as shown in Fig. 6; the extracted motion is displayed on the host PC in real time.

The *Block Module* is shown in Fig. 7. It is a 16-bit processor with 12-bit instruction code consisting of load/store operations, memory read/write operations, logical operations, addition, subtraction, and transfer operations between adjacent *Block Modules*. Although all *Block Modules* execute the same instruction, they have a function for executing different processing depending on their data. In order to realize this function, a *z-flag* and *x-flag* are attached to each instruction. Moreover, a one-bit *z-register Z* is introduced into the *Block Module*. If the result of the previous calculation is

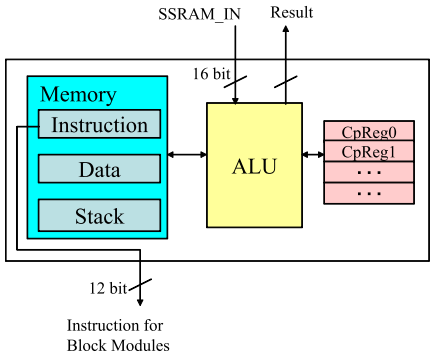


Fig. 8 Block diagram of *Control Processor*.

zero, the *z-register Z* is set. When the *z-flag* of an instruction is set, the instruction is executed only if $Z = 1$. By using the *Halt* instruction, *Block Modules* enter the *Halt state*, where all instructions are ignored. Block modules in the *Halt state* can be restarted by the *Restart* instruction. Even when some *Block Modules* are in the halt state, data transfer must be executed in all *Block Modules*. If the *x-flag* of an instruction is set, the *Block Module* in the *Halt state* executes the instruction. We determined that the width of buses connecting adjacent *Block Module* was 2 bits according to the relation between the processing and the data transfer speed.

The *Control Processor* is shown in Fig. 8. It is a 16-bit processor with 16-bit instruction code, and it has an instruction, stack, and data memory. The instruction code can be classified into code for the *Control Processor* and code for the *Block Modules*. All operations are executed within one clock cycle. In addition to the same operations as the *Block Module*, the *Control Processor* has push/pop and branch operations. None of our processors has a multiplier or divider. Real-time tracking of a moving object is realized by combining simple operations, such as addition and subtraction.

3.2 Synchronous Shift Data Transfer

In this subsection, we describe a *synchronous shift data transfer (SSDT)* technique that allows each *Block Module* to communicate with its neighboring *Block Modules*. Let r be the reference range of neighboring *Block Modules*; each *Block Module* receives data from its $(2r + 1)^2 - 1$ neighboring *Block Modules*. The value of r differs from processing to processing. For the similarity calculation, r is set as the maximum moving range of object. For the object ID estimation, r is set as 1 to receive data of its eight neighboring *Block Modules*. SSDT is based on the following concept. First, if each *Block Module* communicates with a *Block Module* located at the identical relative position, memory contention can be avoided. Second, we can save hardware resources by limiting the network to the interconnections between adjacent *Block Modules*. Third, even under this limitation, each *Block Module* can communicate with any *Block Module* by iteratively transferring data in four directions.

Each *Block Module* has a data transfer module, which is connected to only four neighboring modules to its left,

Table 1 Comparison of direct connections and synchoronous shift data transfer.

	Direct connections		Synchronous shift data transfer	
	Equation	300 modules, 8 bit, $r = 3$	Equation	300 modules, 8 bit, $r = 3$
No. of bus sets/module	$(2r+1)^2 - 1$	48	4	4
No. of total bus sets	$((2r+1)^2 - 1)N$	14400	$4N$	1200
No. of total wires	$((2r+1)^2 - 1)Nb$	115200	$4Nb$	9600
Wire length/module	$L\sum_{i=-r}^{r}\sum_{j=-r}^{r}(\|i\| + \|j\|)b$	$1344L$	$4Lb$	$32L$
Latency	S	S	$S((2r+1)^2 - 1)$	$48S$
Execution time	$T(2r+1)^2$	$49T$	$\max\{S, T\}(2r+1)^2$	$49T$ (if $S \le T$)

r: reference range
N: number of PEs
S: data transfer time for a unit of data
L: length of wire between adjacent PEs
b: bit width of bus line
T: unit data execution time

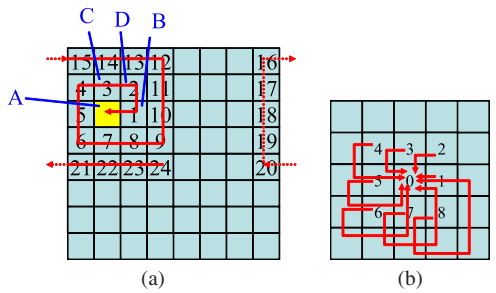


Fig. 9 Data transfer order of synchronous shift data transfer. (a) Data transfer order for communicating with 24 neighboring *Block Modules*. (b) Data Transfer routes of eight neighboring *Block Modules*.

right, top, and bottom, as shown in Fig. 6. The data transfer module transfers unit data for processing. Moreover, a module has a buffer for unit data, in which the received data is stored. In SSDT, all *Block Modules* transfer data in an identical direction simultaneously. By applying SSDT iteratively, each *Block Module* can receive data from its neighboring *Block Module* in a certain order.

In Fig. 9 (a), we show the order of data received from 24 neighboring *Block Modules*. Each square represents a *Block Module*, a red arrow represents the data transfer direction, and the number assigned to each *Block Module* represents the order in which *Block Module* A receives data from the neighboring *Block Module*. For examples, first, SSDT is executed in the left direction. *Block Module* A receives data from the right *Block Module* B. At the same time, *Block Module* C receives data from the right *Block Module* D. Next, SSDT is executed in the bottom direction. *Block Module* A receives data from the upper *Block Module* C, which is initially located at *Block Module* D. Thus, each *Block Module* can receive data from its neighbor *Block Modules*. Figure 9 (b) shows the transfer routes by which data of the eight neighboring blocks arrive at *Block Module* A. Processing with neighboring *Block Modules* is performed not only for *Block Module* A but also for the other *Block Modules*. Since each data is used by the *Block Module* on its transfer route through iterative SSDTs, no redundant transfers occur.

In Table 1, we show a comparison of a two-dimensional array processor based on SSDT with an ideal two-dimensional array processor equipped with

1. Connections between each PE and its $(2r+1)^2-1$ neighbor PEs,
2. Memory with infinite bandwidth.

This ideal system cannot be implemented in reality since it wastes too many hardware resources, and memory access is limited. By controlling data transfer in the two-dimensional array processor in the manner of SSDT, even a systolic array-based architecture can achieve a sufficient performance level which is comparable to the ideal system if $S \le T$, where S is the time taken for unit data transfer between PEs, and T is the unit data execution time.

In the similarity calculation, the data transferred over the boundary of a two-dimensional array should not be used. In order to handle this calculation, *Horizontal-out flag (H-out flag)* and *Vertical-out flag (V-out flag)* are attached to each data.

3.3 Savings-Regret Approximation for Linear Assignment Problem

In order to determine the correspondence between blocks, we solve the linear assignment problem shown in Fig. 2. It is known that the time taken to obtain the optimal solution of a linear assignment problem is $O(n^3)$ [17]. However, this is not suitable for parallel computing. In this paper, we adopt savings-regret approximation [15] as an approximate solver of a linear assignment problem.

We briefly explain the savings-regret approximation. Let $\mathtt{min}_k(i)$ be the k-th smallest element in the i-th row. If all $\mathtt{min}_1(i)$ can be selected for each i-th row from different columns, apparently, the optimal solution is obtained. When the minimum values in different rows are in the same columns, we cannot select both of them. For each row i, we estimate the regret of selecting $\mathtt{min}_k(i)$ instead of $\mathtt{min}_1(i)$, which is denoted by $\mathtt{Diff}_{k-1}$; that is, $\mathtt{Diff}_{k-1}(\mathtt{i}) = \mathtt{min}_k(i) - \mathtt{min}_1(i)$ $(k \ge 2)$.

A solution is obtained through the following steps.

Step 1. Calculate $\mathtt{Diff}_1$ for all uncovered rows in which no element is selected.

Step 2. From the uncovered rows, find the (s, t)-element

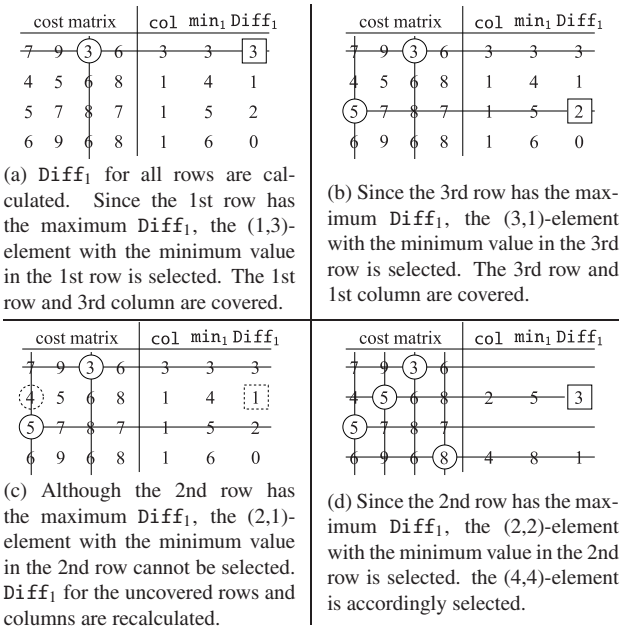


Fig. 10 Example of flow of savings-regret approximation.

such that $s = \arg\max_i \{\mathtt{Diff}_1(i)\}$ and t is the column number of $min_1(i)$.

Step 3. If the t-th column is uncovered, select the (s,t)-element and cover the s-th row and t-th column; otherwise, go to step 1.

Step 4. If there exists an uncovered row, go to step 2; otherwise, terminate.

An example of the flow of the savings-regret approximation is shown in Fig. 10, where `col` represents the column number of the minimum cost for each row. In our hardware, if multiple rows are tied in terms of $\mathtt{Diff}_1(\cdot)$, we also compare $\mathtt{Diff}_2(\cdot)$. When both $\mathtt{Diff}_1(\cdot)$ and $\mathtt{Diff}_2(\cdot)$ have the same value, we use their column numbers for tie breaking.

In order to accelerate the savings-regret approximation, we introduce two special modules into each *Block Module*. The first special unit is *MinPos*, which calculates $\mathtt{Diff}_1$, $\mathtt{Diff}_2$, and `col`. The *MinPos* circuit is shown in Fig. 11. The inputs of this circuit are two values: (a) the 16-bit cost between the own current frame's block and a received previous frame's block; and (b) the column number corresponding to the module ID of the previous frame's block. $\mathtt{min}_1, \mathtt{min}_2$, and $\mathtt{min}_3$ are updated while their current values are compared with the input cost. After all costs in a row of the cost matrix have been compared, $\mathtt{Diff}_1$, $\mathtt{Diff}_2$, and `col` are calculated. We employed bit-serial comparators to reduce the size of the circuit. Step 1 is performed for each row in parallel by using *MinPos* of the corresponding *Block Module*. Let R_{obj} be the maximum move range of object. It was set as 3 in our implementation. Each row has $(2R_{obj}+1)^2+2$ elements: the cost for $(2R_{obj}+1)^2$ neighboring previous frame's blocks, background, and create. This

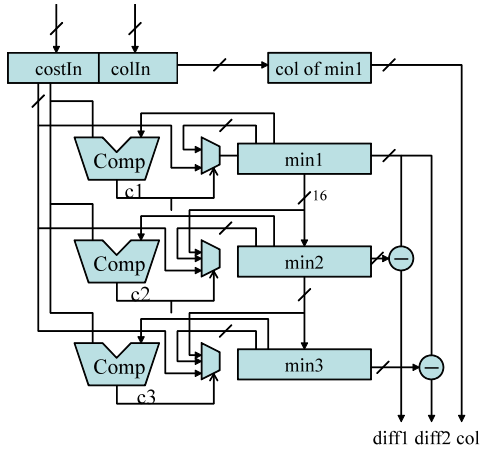


Fig. 11 MinPos module.

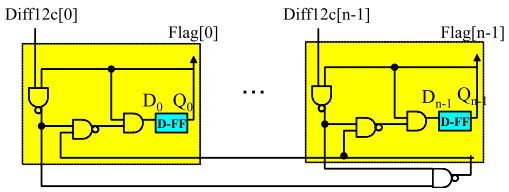


Fig. 12 SelMax module.

step takes $O(R_{obj}^2)$.

The second special unit for step 2 of savings-regret approximation is *SelMax*, shown in Fig. 12. The unit is based on a Winner-Take-All (WTA) circuit [16], and the input $\mathtt{Diff}_{12c}$ is the 48-bit concatenated data of $\mathtt{Diff}_1$, $\mathtt{Diff}_2$, and `col`. *SelMax* compares $\mathtt{Diff}_{12c}$ of all the *Block Modules*, and set the output flag of the *Block Module* that has the maximum concatenated data. Step 2 can be executed in $O(1)$ by using *SelMax* module.

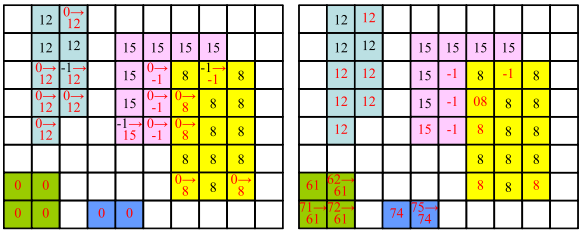
According to step 3 and step 4, step 1 and step 2 are iterated at most n times where n is the number of rows (*Block Modules*). Therefore, our hardware can obtain an exclusive block matching in $O(R_{obj}^2 n)$. Since R_{obj} can be regarded as a small constant number, we can obtain an exclusive block matching approximately in $O(n)$.

3.4 Object ID Estimation

Each block has two kinds of values: objBg, and objID. objBg indicates whether or not the block is background. objID is the identification code of each object.

After solving the linear assignment problem shown in Fig. 2, the current frame's block is regarded as the object block if it is matched with a previous frame's block or a created block. Each object block is assigned to an object ID in the following way.

If a current frame's block is matched with a previous frame's block, the current frame's block inherits the objID from the previous frame's block. The remaining blocks are *undefined* or *created blocks*. Their objIDs are determined by their eight neighboring *Block Modules* as follows. (a) If



(a) (b)

Fig. 13 Object ID estimation.

there is only one objID in the neighboring blocks, the objID is assigned. (b) If there are more than one objID, the objID is *undefined*. The above operation for each block is iteratively performed in parallel while there is an updated objID. An example of this processing is shown in Fig. 13 (a).

The remaining blocks are composed of only newly *created blocks*; they have a label of zero. Their labels are updated according to the connected component labeling. First, the blocks labeled as zero are set as the module ID of their *Block Module*. Then, each *Block Module* receives the labels of the eight neighboring blocks, and updates its own label with the minimum label among them. This processing is iteratively executed for each block in parallel while there exists an updated label. As a result, the blocks of an identical object are assigned to the same label, and those of different objects are assigned to different labels. An example of object ID estimation based on connected-component labeling is shown in Fig. 13 (b).

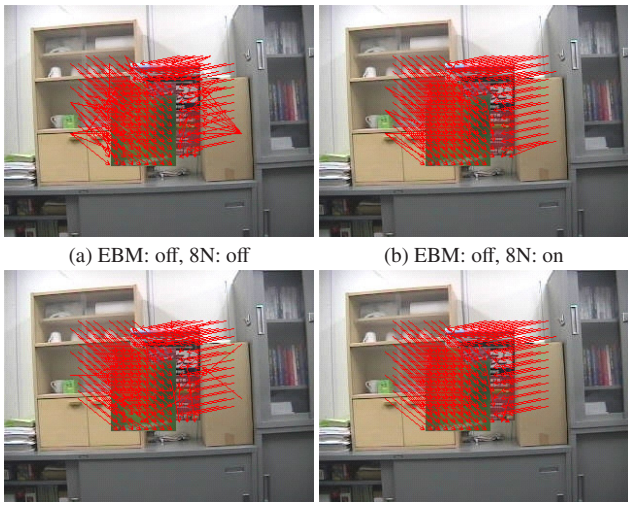
3.5 Background Update

The background is calculated using IIR filtering. Since the *Block Modules* have only histograms of image data, the background is calculated by IIR filtering of histograms. The results of IIR filtering of histograms are almost same as those of IIR filtering of the original image data.

4. Experimental Results

4.1 Evaluation of Exclusive Block Matching and Consideration of the Eight Neighboring Blocks

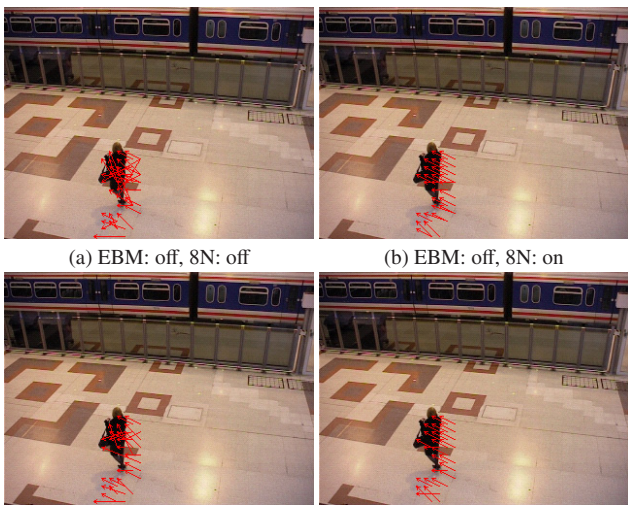
Figure 14 (a) and (b) shows the results when the matching with the minimum cost is selected from each row; the matching is not exclusive. Thus, there exist multiple flows incident from an identical block. On the other hand, Fig. 14 (c) and (d) shows the results when the exclusive correspondences of blocks are obtained by saving-regrets approximation. Moreover, in Fig. 14 (b) and (d), the costs of the eight neighboring blocks are reflected, as described in Sect. 2.4. By considering the cost of eight neighbor blocks, the flow disturbance is reduced considerably. Similarly, the results for PETS2006 benchmark data [18] and an indoor scene are shown in Fig. 15 and Fig. 16, respectively. In Fig. 16, the walls are plain and almost monochrome, and the flow seen in Fig. 16 (a) is disturbed. It can be seen in



(a) EBM: off, 8N: off (b) EBM: off, 8N: on

(c) EBM: on, 8N: off (d) EBM: on, 8N: on

Fig. 14 Effects of exclusive block matching (EBM) and consideration of eight neighboring blocks (8N) for CG data.



(a) EBM: off, 8N: off (b) EBM: off, 8N: on

(c) EBM: on, 8N: off (d) EBM: on, 8N: on

Fig. 15 Effects of exclusive block matching (EBM) and consideration of eight neighboring blocks (8N) for PETS2006 benchmark data.

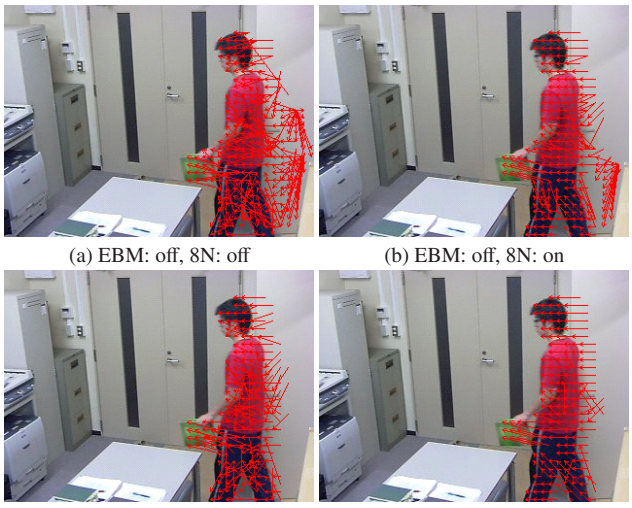
Fig. 16 (d) that the flow is improved by the effects of both exclusive correspondences and considering eight neighboring blocks.

4.2 Comparison with Dense SIFT and Software Implementation

First, we show the comparison between the results of the EBM method with dense SIFT features and those with HSV and HOG features. We calculated dense SIFT features for each block in the following manner. A sampling point was generated at the center of each sub-block of 4×4 pixels. For each sampling point, we considered three different scales: $\sigma = 8$, 10, or 13 pixels. Thus, we calculated 12 SIFT features for each block. The cost between blocks a and b is calculated by $\min_{f_a \in \mathbf{F}_a, f_b \in \mathbf{F}_b} C_8(f_a, f_b)$ where $\mathbf{F}_a$ and $\mathbf{F}_b$ are

Table 2 Comparison of the ratio of the inaccuracy motions.

	Feature	Assignment	Distance	average	Indoor scene in Fig. 16	PETS2006 [18]	Shopping Center [19]
	Dense SIFT	optimal	Bhattacharyya	4.19%	5.41%	3.52%	3.64%
software implementation	HSV+HOG	optimal	Bhattacharyya	4.43%	4.72%	5.97%	2.59%
	HSV+HOG	optimal	absolute difference	5.50%	5.54%	7.50%	3.47%
hardware implementation	HSV+HOG	savings-regret	absolute difference	6.81%	8.00%	8.46%	3.98%



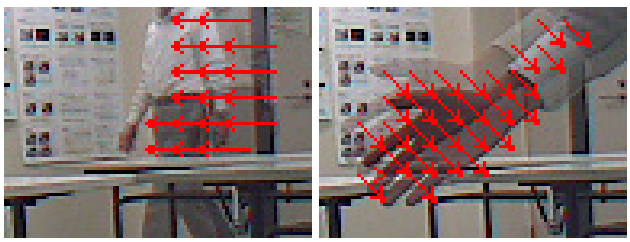
(a) EBM: off, 8N: off (b) EBM: off, 8N: on

(c) EBM: on, 8N: off (d) EBM: on, 8N: on

Fig. 16 Effects of exclusive block matching (EBM) and consideration of eight neighboring blocks (8N) for real data.

Table 3 The logic utilization of EBM hardware for each number of *Block Modules*.

No. of *Block Modules*	No. of ALUTs	No. of registers
108	65983	53614
300	172982	145528
768	393678	367136



(a) (b)

Fig. 17 Examples of the outputs of object tracking hardware when the array size is 12×9.

the sets of 12 SIFT features of block a and b, respectively.

For these experiments, we used three data sets: indoor scene shown in Fig. 16, PETS2006 benchmark data, and shopping mall scene [19]. We calculated the ratio of the inaccurate motions which was visually determined with considering unnatural crossing [20]. The first row and the second row of Table 2 show the results of the EBM method with dense SIFT features and those with HSV and HOG features. We can see that the accuracy of extracted motion in employing HSV and HOG features is comparable with that in employing dense SIFT features even though the calculation of HSV and HOG feature is simpler than dense SIFT features.

Next, we show that the comparison between the results of software implementation and that with hardware implementation. In order to save hardware resources and realize highly parallel computations, our hardware employs the sum of absolute difference and savings-regret approximation instead of Bhattacharyya distance and the method to obtain the optimal assignment. We evaluated the effects of these simplifications. The third row of Table 2 shows the results in replacing Bhattacharyya distance with the sum of absolute difference. The fourth row shows the results of hardware implementation which employs the sum of absolute difference and savings-regret approximation. The degradation arising from the simplifications for hardware implementation was only 2.38% on an average.

4.3 Implementation Results of Tracking Hardware

We implemented the proposed tracking hardware in an Altera stratix III (EP3SL150F780C4) device, and validated the object tracking. For object tracking, we could implement 108 (12×9) *Block Modules* on the device. The maximum operating frequency was 48.01 MHz. We wrote the program in C-like code and used assembler code for the array processor. The results of object tracking for real data are shown in Fig. 17.

The logic utilization for object tracking is shown in Table 3. In Table 3, we also present the results of compilation on a high-end Altera stratix V (5SEEBH40C2L) device to show the logic utilization for a larger number of *Block Modules*. We can easily increase the number of *Block Modules* until the resources of the device are exhausted while maintaining the performance of each *Block Module*. The amount of logic utilization increases almost proportionally to the number of *Block Modules*. The ALUT utilization is approximated by $492N_{BM}+18076$, where N_{BM} is the number of *Block Modules*.

4.4 Performance Estimation of Tracking Hardware

We developed an emulator of our hardware, and we evaluated the performance of hardware equipped with a larger number of *Block Modules*. Table 4 lists the number of clock cycles required for each operation in object tracking. Since

Table 4 Clock cycles for each operation in object tracking.

No. of *Block Modules*	108	300	768	1200
Image size	96×72	160×120	256×196	320×240
Similarity calculation	159053	159053	159053	159053
Assignment	13248	34440	84540	133747
Object ID estimation	1549	1549	1598	1722
Background update	1446	1446	1446	1446
Result send	350	926	2330	3626
Etc.	12683	12683	12683	12683
No. of total steps	188329	210121	261650	312277
fps at 48 MHz clock	267	240	192	161

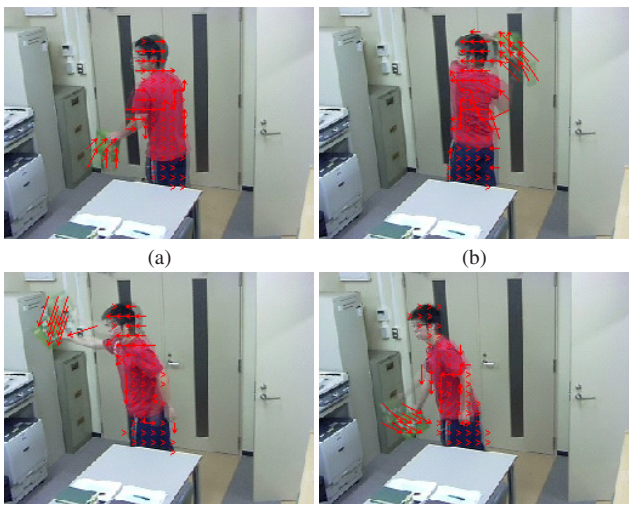


Fig. 18 Example of the outputs of emulator of object tracking hardware when the array size is 32×24.

they vary according to the scene of the assignment and object separation, we calculate the average for fifteen frames from PETS2006 benchmark data. Even if the image size increases, the number of clock cycles required for similarity calculation and background update remains constant. This is because the effective number of PEs for similarity calculations increases in proportion to the image size. Although the number of clock cycles for savings-regret approximation depends on the scene, in practice, the number of clock cycles for an assignment is almost proportional to the image size. The object ID estimation, background update, and result send constitute a small fraction of the total clock cycles. The remaining operations are control operations executed by the *Control Processor*, and the number of clock cycles is constant.

By exploiting the high parallelism of the object tracking algorithm, our array processor can treat each processing effectively. The data transfer from *HSVGen*/*HOGGen* to *Block Modules*, shown in Fig. 6, is performed simultaneously with tracking by using two-port memory. Therefore, the transfer time for each frame data does not affect the performance of the hardware provided that the data transfer is finished within the processing time for a frame. When we implement 1200 *Block Modules*, we can handle a QVGA image by dividing it into blocks of 8×8 pixels. When the system clock frequency is 48 MHz, this circuit realizes object tracking at 161 fps for QVGA image sequences. It is about 53 times faster than that for executing EBM method on 3.14 GHz CPU. Figure 18 shows examples of motion extraction of a a moving object; the extracted motions could be useful for human motion analysis.

5. Conclusions

In this paper, we presented an implementation of an exclusive block matching method that realizes detailed motion extraction and robust tracking. Our tracking hardware can realize robust moving object extraction and tracking at more than 100 fps for QVGA images. The current hardware is designed for processing scenes captured with a fixed camera. Enhancement of scenes captured with a motion camera will be addressed in our future studies.

References

[1] K. Mizuno, H. Noguchi, G. He, Y. Terachi, T. Kamino, T. Fujinaga, S. Izumi, Y. Ariki, H. Kawaguchi, and M. Yoshimoto, "A low-power real-time SIFT descriptor generation engine for full-HDTV video recognition," IEICE Trans. Electron., vol.E94-C, no.4, pp.448–457, April 2011.
[2] F.C. Huang, S.Y. Huang, J.W. Ker, and Y.C. Chen, "High-performance sift hardware accelerator for real-time image feature extraction," IEEE Trans. Circuits Syst. Video Technol., vol.22, no.3, pp.340–351, March 2012.
[3] K. Mizuno, Y. Terachi, K. Takagi, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "An FPGA implementation of a HOG-based object detection processor," IPSJ Transactions on System LSI Design Methodology, vol.6, pp.42–51, Feb. 2013.
[4] B.K.P. Horn and B.G. Schunck, "Determining optical flow," ARTIFICAL INTELLIGENCE, vol.17, pp.185–203, 1981.
[5] J.Y. Bouguet, Pyramidal implementation of the lucas kanade feature tracker, Intel Corporation, Microprocessor Research Labs, 2000.
[6] D. Lowe, "Object recognition from local scale-invariant features," Proceedings of the International Conference on Computer Vision 2, pp.1150–1157, 1999.
[7] H. Bay, T. Tuytelaars, and L.V. Gool, "Surf: Speeded up robust features," European Conference on Computer Vision, pp.404–417, 2006.
[8] Z. Li, K. Yabuta, and H. Kitazawa, "Exclusive block matching for moving object extraction and tracking," IEICE Trans. Inf. & Syst., vol.E93-D, no.5, pp.1263–1271, May 2010.
[9] D. Comaniciu, P. Meer, and S. Member, "Mean shift: A robust approach toward feature space analysis," IEEE Trans. Pattern Anal. Mach. Intell., vol.24, pp.603–619, 2002.
[10] P. Pe'rez, C. Hue, J. Vermaak, and M. Gangnet, "Color-based probabilistic tracking," European Conference on Computer Vision, pp.661–675, 2002.
[11] K. Nummiaro, E. Koller-meier, and L.V. Gool, "A color-based particle filter," Image and Vision Computing, pp.53–60, 2002.
[12] I. Ishii, T. Taniguchi, K. Yamamoto, and T. Takaki, "High-frame-rate optical flow system," IEEE Trans. Circuits Syst. Video Technol., vol.22, no.1, pp.105–112, Jan. 2012.
[13] G.K. Gultekin and A. Saranli, "An FPGA based high performance optical flow hardware design for computer vision applications," Microprocessors and Microsystems, vol.37, no.3, pp.270–286, 2013.
[14] E. Norouznezhad, A. Bigdeli, A. Postula, and B. Lovell, "Robust object tracking using local oriented energy features and its hardware/software implementation," Control Automation Robotics Vi-

sion (ICARCV), 2010 11th International Conference on, pp.2060–2066, Dec. 2010.
[15] M.A. Trick, "A linear relaxation heuristic for the generalized assignment problem," Naval Research Logistics, vol.39, pp.137–151, 1992.
[16] H. Tamukoh, K. Horio, and T. Yamakawa, "Fast learning algorithms for self-organizing map employing rough comparison wta and its digital hardware implementation," IEICE Trans. Electron., vol.E87-C, no.11, pp.1787–1794, Nov. 2004.
[17] R. Jonker and A. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems," Computing, vol.38, no.4, pp.325–340, Nov. 1987.
[18] "PETS2006 benchmark data," http://www.cvg.rdg.ac.uk/PETS2006/data.html
[19] "CAVIAR test case scenarios," http://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1/
[20] Z. Li, Y. Tomioka, and H. Kitazawa, "Extraction and tracking moving objects in detail considering visual feature constraint and structure constraint," IEICE Trans. Inf. & Syst., vol.E96-D, no.5, pp.1171–1181, May 2013.

Yoichi Tomioka received his B.E., M.E., and D.E. degrees from Tokyo Institute of Technology, Tokyo, Japan, in 2005, 2006, and 2009, respectively. He was a research associate at Tokyo Institute of Technology till 2009. Since 2009, he has been an assistant professor in the Division of Advanced Electrical and Electronics Engineering at Tokyo University of Agriculture and Technology. His research interests include image processing, security systems with mobile robots, VLSI package design automation, and combinational algorithms. He is a member of IEEE and IPSJ.

Ryota Takasu received his B.S. degree in Electrical and Electronic Engineering from Tokyo University of Agriculture and Technology, Japan, in 2013. He is currently pursuing a master's course at the university. His research interests include image processing.

Takashi Aoki received the B.S. and M.S. degrees in applied physics from Tokyo Institute of Technology, Tokyo, in 1987 and 1989, respectively. He joined NTT in 1989 and studied LSI design and network design. He is currently a Research Engineer at NTT Microsystem Integration Laboratories. He is a member of the IPSJ, and the Physical Society of Japan.

Eiichi Hosoya received his B.E. in 1991 and M.E. in 1993 both in electrical and electronics engineering from Chiba University, Japan, and Ph.D. in 2011 in Information Science from Nagoya University, Japan. He joined NTT LSI Laboratories, Kanagawa, in 1993. He has been working for the research and development of LSI design technologies, computer vision and human computer interaction. He is currently a Senior Research Engineer, Supervisor at NTT Microsystem Integration Laboratories. He is a member of the IPSJ.

Hitoshi Kitazawa received his B.S., M.S., and Ph.D. degrees in Electronic Engineering from Tokyo Institute of Technology, Tokyo, Japan, in 1974, 1976 and 1979, respectively. He joined the Electrical Communication Laboratories, Nippon Telegraph and Telephone Corporation (NTT), in 1979. Since 2002, he is a professor at Tokyo University of Agriculture and Technology. His research interests include VLSI CAD algorithm, Computer Graphics and Image Processing. He is a member of IPSJ and IEEE.