Journal of Signal Processing, Vol.18, No.4, pp.165-168, July 2014

SELECTED PAPER AT NCSP'14

FPGA-Based Design for Motion Vector Estimation Exploiting High-Speed Imaging and Its Application to Motion Classification with Neural Networks

Masafumi Mori[1], Toshiyuki Itou[1] , Masayuki Ikebe[1], Tetsuya Asai[1], Tadahiro Kuroda[2] and Masato Motomura[1]

[1]Graduate School of IST, Hokkaido University
Kita 14, Nishi 9, Kita–ku, Sapporo 060–0814, Japan
E-mail: mori@lalsie.ist.hokudai.ac.jp, asai@ist.hokudai.ac.jp

[2]Faculty of Science and Technology, Keio University
Hiyoshi 3–14–1, Kohoku-ku, Yokohama,
Kanagawa 223–8522, Japan

Abstract

In this study, we propose an architecture for estimating motion vectors by searching for one neighbor pixel in high-speed images and a machine learning algorithm that uses the estimated motion vectors. In high-speed imaging, the motion of pixels between frames is considerably small. Our architecture estimates motion vectors by assuming that the pixels move less than one pixel between frames. We verified that our method could classify images into two classes, i.e., dangerous (something is approaching) or safe (others), by employing a simple perceptron after extracting the features of the estimated motion vectors using a method based on Poggio's HMAX (Hierarchical Model and X) model. We used the target images captured by an in-vehicle camera for learning and verified that another set of images could be classified using our method. We confirmed that the proposed architecture can estimate motion vectors using a small number of operations and perform classification based on machine learning.

1. Introduction

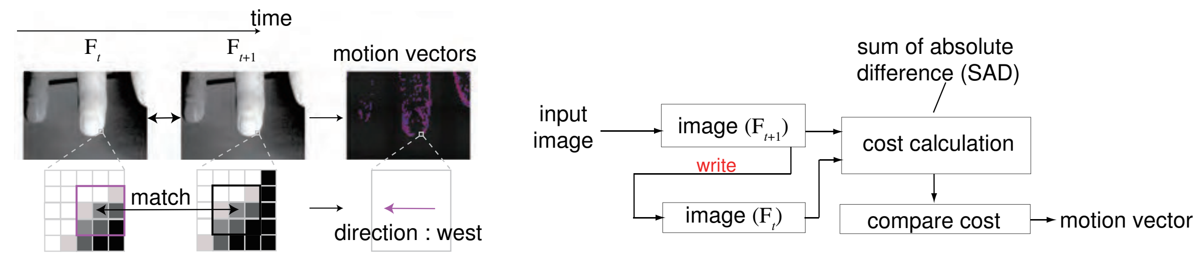
Image processors that can be installed in portable terminals such as smartphones are being actively developed [1]. Traditionally, architectures used for complex processing consisted of multiple connected processors or memory components on a printed circuit board. However, this led to a significant increase in the size of the board, and a low rate of data transmission between devices [2]. Thus, a method was devised that integrated chips and connected them in three dimensions [3]. Using this method, the chip area was reduced and the data rate between chips improved. We expect that the data rate will be considerably increased in the near future, opening up a new field of modern semiconductor applications. For example, we may assume that images captured by an image sensor at 1,000 fps will be directly transmitted to an image processor. In this case, the interframe differences become smaller as the video frame rate increases, which means that the search ranges used for motion vector estimation by block matching decreases. Therefore, motion vectors can be estimated using a small number of calculations. In the present study, we aimed to apply machine learning to motion vectors. Therefore, we devised a method that allows motion vectors, to be estimated using our proposed architecture and their use for machine learning.

In this study, we developed an architecture for estimating motion vectors based on a small number of calculations. Our architecture estimates motion vectors by assuming that a pixel at specific coordinates moves less than one pixel between frames. Furthermore, we propose a method that classifies images into two classes, *i.e.*, dangerous (an object is approaching) or safe (others), using a simple perceptron, after extracting the features of the motion vectors estimated by the architecture using a method based on Poggio's HMAX (Hierarchical Model and X) model [4]. We verified that machine learning was possible using our feature extraction method because a simple perceptron can only classify linearly separable problems. Therefore, we used target images captured with a high-speed camera or an in-vehicle camera for learning, and then we verified that the same or similar images could be classified with our proposed method.

2. Motion Vector Estimation for High-Speed Imaging

2.1 Proposed algorithm

In addition to conventional optic-flow based motion estimation [5], a block-matching method can be used as a practical algorithm for detecting motion in moving images [6]. This block-matching method requires many calculations, resulting in a long execution time. To reduce the processing time of the block-matching method, it is necessary to reduce the search range. However, the block-matching method has low accuracy if the search range is reduced in advance. In high-speed imaging, the motion of a pixel at specific coordinates becomes considerably small between frames. In the present study, therefore, we estimate motion vectors by assuming i) the use of high-speed video cameras and, conse-



(a) Image of division for block matching (b) Proposed algorithm for motion vector estimation

Figure 1: Process flow of proposed algorithm for motion vector estimation

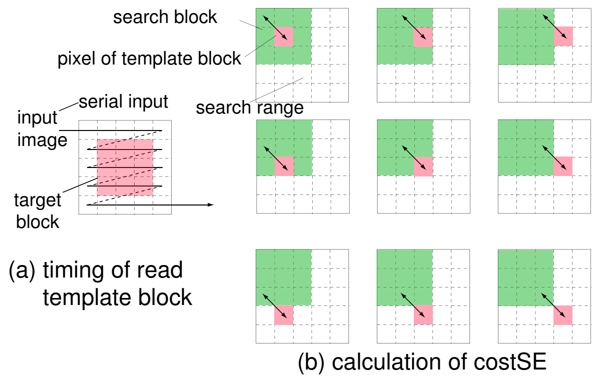


Figure 2: Sequential cost calculation timings

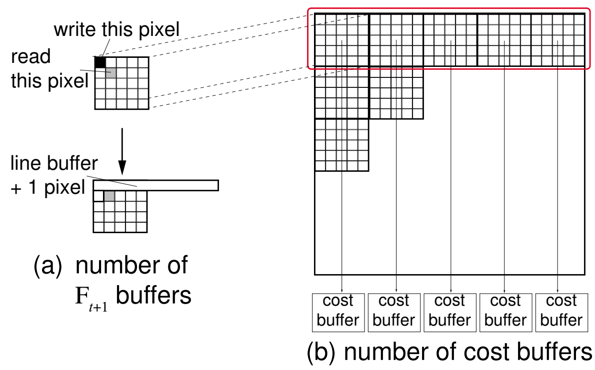


Figure 3: Estimated number of F_{t+1} buffers

quently, ii) pixels move less than one pixel between frames. We set the nearest-neighbor pixel in the target block as the search range and then estimate the motion vector.

Figure 1(a) shows how the search range is divided for images. We define F_t and F_{t+1} as two consecutive frames in time. The search range is 5×5 pixels. We divide the images into search ranges and motion vector estimation is performed for each search range, where the block size is 3×3 pixels. The target block is a center search block with a search range of F_{t+1}. The search blocks cover eight directions by moving one pixel from the center block in the search range. Our architecture estimates the match between a target block and a search block in the search range. The sum of absolute differences (SAD) is used to determine a suitable motion vector. We denote the SAD as cost{DIRECTION}, for example, costNW represents the cost between the northwest block and the center block, while costN represents the cost between the north block and the center block. Our block-matching method is applied by determining the minimum cost. Figure 1(b) shows the motion vector estimation algorithm. Our architecture calculates the costs using the pixels from both frames. To facilitate real-time processing, our architecture rewrites a pixel from F_{t+1} sequentially after calculating the costs using a pixel from F_t. The number of SAD calculations is decreased by reducing the search range. Our architecture can estimate the motion vectors within a short period of time.

2.2 Proposed architecture

We here explain the architecture used for motion vector estimation. Figure 2(a) shows the timing when pixels in the target block are read. Note that the inputs always flow to outputs in a straightforward manner in this model. Figure 2(b) shows the process used to calculate costNW. CostNW is the SAD for a target block and the search block located northwest of the center of the search block. When a target block pixel is inputted, our architecture calculates the sum of the absolute difference between the pixels in the target block and the pixels in the corresponding search block. Our architecture simultaneously calculates the SADs in eight directions for the other search blocks.

Figure 3(a) shows that several registers are required to rewrite the frame F_t. If the gray pixels are used as inputs, the black pixels are not used as search blocks after calculating the costs. Our architecture rewrites the pixels. The buffers required to hold the pixels for F_{t+1} contain the line buffer and one pixel. Figure 3(b) shows the number of buffers needed to store the intermediate results of the cost calculations. The images are input in series. The pixels in an image are used as inputs in sequence across the search range. During the cal-

Table 1: FPGA implementation summary

Input Res.	Depth	CLK	LUT	Reg.	Block mem.
10×10	16-bit	80 MHz	1053	572	12736 bits

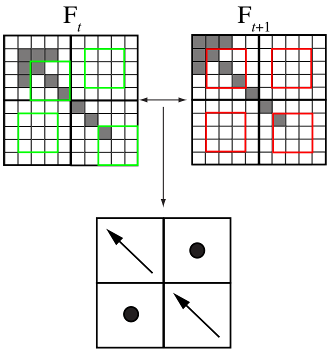


Figure 4: Schematic image of motion vector estimation

culation of a cost in a search range, our architecture needs to hold costs in another search range. The cost calculations for a search range require nine buffers to hold the intermediate cost calculations. The total number of required buffers for the cost calculation is the number of blocks per line $\times$ 9 [$= 5 \times 9 = 45$, in case of an example shown in Fig. 3(b)].

2.3 FPGA implementation results

We analyzed the results obtained using the proposed methods based on the field-programmable gate array (FPGA) implementation of the motion vector estimation architecture. The proposed system for motion vector estimation was implemented using a commercial FPGA board (MU-200SX II with Altera Stratix II). Table 1 summarizes the implementation setup. The input images contained 10 $\times$ 10 pixels. The timing clock of the architecture was operated at 80 MHz, which is the highest operating clock speed for a MU-200SX II system. We verified that our architecture could operate at a clock speed of 80 MHz. Figure 4 shows one of the test patterns used for motion vector estimation. We assumed that the pixels moved less than one pixel between frames. We show the results of generating an image from the output signal of the FPGA. We confirmed that the desired results were obtained.

3. Machine Learning of Motion Vectors

Figure 5 shows a summary of the proposed machine learning method used for motion vectors. Images must be subjected to feature extraction by machine learning before they can be used as inputs by a neural network [7]. An image is separated into specific block sizes and the direction and size of each vector are extracted, as shown in Fig. 5(a). Next,

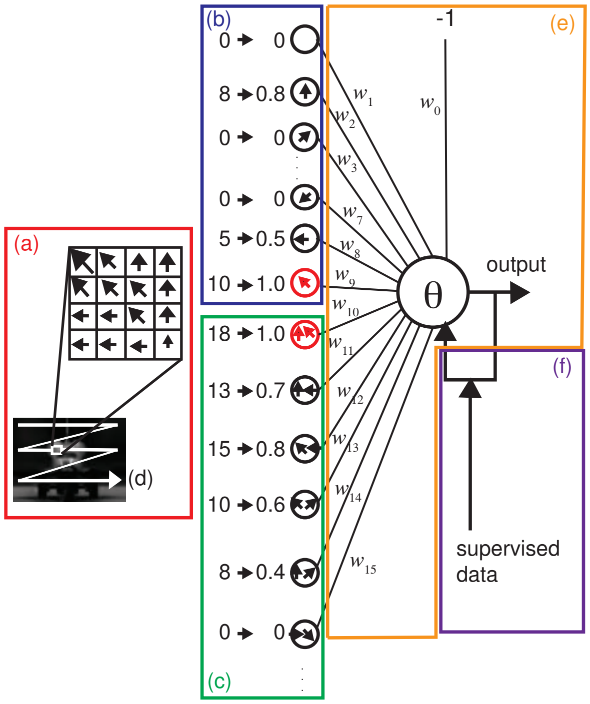


Figure 5: Summary of machine learning process for motion vectors

the sums of the vector sizes in a block are calculated in each direction and the values are normalized to values from 0 to 1, as shown in Fig. 5(b). Furthermore, the summed vector sizes are calculated for each combination of two vectors and the values are normalized to values of 0 to 1, as shown in Fig. 5(c). All blocks are processed in a similar manner, as shown in Fig. 5(d). This feature extraction method is based on Poggio's HMAX model [4]. Next, the set of values is used as an input for a simple perceptron. In the simple perceptron, the values are connected by weights (from w_1 to w_{15} in Fig. 5), where the output is 1 if the sum of the values exceeds a threshold value (w_0 in Fig. 5), whereas the output is –1 if the sum of the values does not exceed the threshold value, as shown in Fig. 5(e). The user provides supervised data when the simple perceptron is learning, as shown in Fig. 5(f). The simple perceptron compares the output with the supervised data and the connected weights are updated if the two values are different.

We verified that our method could classify images recorded using a high-speed camera or an in-vehicle camera into two classes, *i.e.*, dangerous or safe, as mentioned earlier. We conducted a simulation where the algorithm was implemented in C language.

First, we used a high-speed camera that operated at 1,000 fps to record various types of motion made by objects (*e.g.*, a space shuttle model or a box), which were verified using their images. The recorded images were used to estimate the

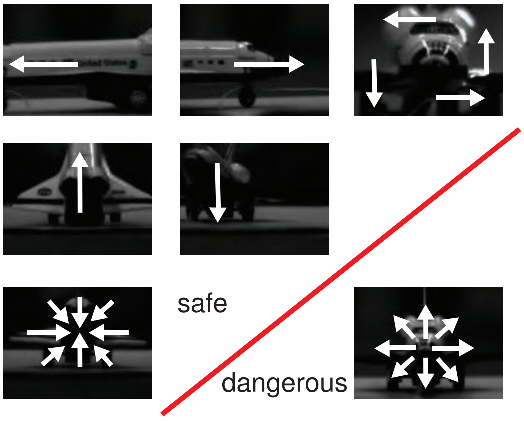


Figure 6: Classification of images recorded using high-speed camera

Figure 7: Classification of images captured using in-vehicle camera

motion vectors, where we assumed that a pixel at specific coordinates moved less than one pixel between frames. Next, we integrated 80 frames of the motion vectors to calculate the vector size. The image size was 400 $\times$ 300 pixels and the block size used was 40 $\times$ 30 pixels. The simple perceptron learned 800 dangerous samples and 1,200 safe samples, and learning converged when the simple perceptron had learned all of the samples approximately 50 times. Figure 6 shows the results for the motion of the space shuttle model, which was recorded using a high-speed camera and classified by the simple perceptron. The samples were used for learning and they were classified with our method.

Furthermore, we verified the suitability of our method using images captured by an in-vehicle camera. For these images, the optical flow was detected as a motion vector by block matching. The image size was 400 $\times$ 300 pixels and the block size used was 40 $\times$ 30 pixels. The simple perceptron learned 212 dangerous samples and 749 safe samples, and learning converged after the simple perceptron had learned all of the samples approximately 500 times. Figure 7 shows the samples recorded using an in-vehicle camera and their classification results with the simple perceptron. The samples were used for learning and they were classified using our method.

4. Summary

In this paper, we reported an ultrasimple architecture for motion vector estimation under high-speed imaging, and proposed a novel motion vector characterization module for a linear classifier (perceptron). In general, the applications of high-speed cameras are limited to making ultraslow-motion pictures, but our approach may open a new application field of high-speed cameras with motion vector processing in a very simple manner. Furthermore, our classification results showed that the linear classifier could discriminate supervised patterns with 100% accuracy, which indicated that the proposed motion characterization unit was able to transform the input motion vectors to linear separation problems successfully.

Acknowledgment

The authors would like to thank the Semiconductor Technology Academic Research Center (STARC) of Japan for funding this research project.

References

[1] T. Onoye: Recent trends on media processors for embedded systems, The Journal of the Institute of Image Information and Television Engineers, Vol. 63, No. 9, pp. 1185–1187, 2009.

[2] R. Sale, S. Wilton, S. Mirabbasi, A. Hu, M. Greenstreet, G. Lemieux, P. Pande, C. Grecu and A. Ivanov: System-on-chip: Reuse and integration, Proc. IEEE, Vol. 94, No. 6, pp. 1050–1069, 2006.

[3] P. Garrou, R. Ramm and C. Bower: Handbook of 3D Integration: Technology and Applications of 3D Integrated Circuits, Wiley-VCH Verlag GmbH, 2008.

[4] M. Risenhumber and T. Poggio: Hierarchical models of object recognition in cortex, Nature Neuroscience, Vol. 2, No. 11, pp. 1019–1025, 1999.

[5] http://en.wikipedia.org/wiki/Optical_flow

[6] D. I. Barnes and H. F. Silverman: A class of algorithm for fast digital image registration, IEEE Trans. on Computers, Vol. 21, No. 2, pp. 179–186, 1972.

[7] I. Guyon: Feature Extraction: Foundations and Applications, Vol. 207, Springer, 2006.