MVA2007 IAPR Conference on Machine Vision Applications, May 16-18, 2007, Tokyo, JAPAN

A Real-Time Hand Gesture Interface Implemented on a Multi-Core Processor

Tsukasa Ike Nobuhisa Kishikawa Björn Stenger
Toshiba Corporation
1, Komukai-Toshiba-cho, Saiwai-ku, Kawasaki, 212-8582, Japan
tsukasa.ike@toshiba.co.jp

Abstract

This paper describes a real-time hand gesture recognition system and its application to VCR remote control. Cascaded classifiers are used to detect a number of different hand poses. In order to detect a hand in real time, the detection algorithm is optimized for multi-core processors by distributing the operations to multiple cores and minimizing the data transmission between them. We have implemented a detection system on a processor with eight cores. Further we have integrated the system into a prototype video recorder simulator to evaluate a gesture interface for consumer electronics. The operating speed increases by a factor of up to 13.5 compared to a standard PC with single-core processor.

1. Introduction

In daily life remote controls are used to activate consumer electronic products such as televisions, video recorders and air-conditioners. However, the number of remote controls and number of different functions of each may be confusing for some users. In such cases gesture control is a viable option for basic operation if the system is able to work in the complex visual scenes which may be present in a living room.

Freeman and Weissman presented a method for TV remote control by hand gestures [2]. They used normalized correlation and edge orientation to detect an open hand. However, the matching cost is high because normalized correlation is calculated for all candidate sub-windows, thus the detection area and the variation of the detection sub-window is restricted. Many systems employ a previously learned skin color model, for example Bretzner et al. [1] used multi-scale color features for hand posture detection. However, determining skin color is difficult because it depends on the lighting condition as well as each individual. Including other cues such as motion and edges [6] can improve the robustness.

Recently cascaded classifiers, originally developed for efficient face detection, have been applied to hand detection [3,5]. No skin color model is required and the method works under a large range of imaging conditions. One difficulty however is that background regions are usually included in the cropped training samples. Kölsch and Turk [3] have shown that such a detector works well in different environments for some hand poses given that the hand shape is well aligned. Ong and Bowden [5] used a classifier hierarchy where hand regions are detected initially and subsequently the pose is determined. The system was reported to perform well in uniform backgrounds.

In this paper we implement the detection method using joint rectangle features [4] for hand detection. This is an extension of Viola and Jones' object detection method [7] but allows for the combination of multiple features in each weak classifier. In [7] the detector is scanned across the image and each sub-window is evaluated using detection cascades which consist of a series of classifiers. The initial classifier rejects the majority of sub-windows with very little processing so that the total computational cost becomes smaller. By using joint features, co-occurrences of multiple features are evaluated, resulting in better classification performance for the same number of features compared to [7]; see [4] for an evaluation on face data.

The operating speed of hand posture detection is still not fast enough to detect multiple hand postures in real time. When using separately trained detectors, one for each pose, the computation cost increases in proportion to the number of detectors. We achieve the target frame rate by optimizing the detection on a multi-core processor. For efficiency we introduce two kinds of optimization methods. First, we modify and divide the operations so that they can be executed efficiently using all cores. Further two kinds of parallel operations for feature computation are introduced. We confirm the efficiency of the method through experiments using a system with a Cell Broadband Engine™, which consists of eight cores. We achieve a speed-up factor of up to 13.5 compared to an implementation on a standard PC with Xeon processor.

2. Prototype hand gesture UI system

Figure 1 shows the prototype hand gesture UI system. It consists of five components; the Cell reference set (CRS: Fig. 1(a)), a display for the recognition result (Fig. 1(b)), a video camera for capturing images (Fig. 1(c)), a PC for running the video recorder simulator (Fig. 1(d)), and a simulator display (Fig. 1(e)).

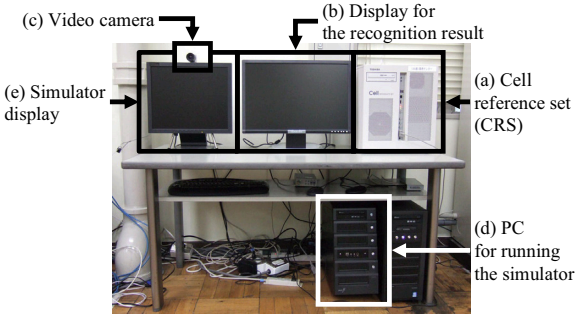
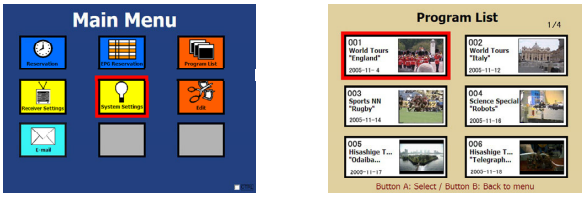


Figure 1. Prototype hand gesture UI system using a multi-core processor

(a) pointing hand (b) fist (c) open hand

Figure 2. Training examples for three different hand poses.



(a) Main menu (b) Program List

Figure 3. VCR simulator menu

Figure 2 shows the three hand poses recognized by the system. When a user holds a hand in front of the camera (Fig. 1(c)), the algorithm running on the CRS recognizes the size, position, and pose of the hand in the image. The detection result is shown on the display (Fig. 1(b)) and the CRS sends a control command to the PC (Fig. 1(d)).

Figure 3 shows the menu screen displayed on the monitor (Fig. 1(e)). On system start-up the main menu (Fig. 3(a)) is shown. The icons correspond to certain commands such as "Program list" or "System settings". When the user makes a pointing gesture (Fig. 2(a)), the cursor (shown as a red rectangle around icons) moves according to the hand motion. The user can execute the commands attached to the each icon by making a fist hand gesture (Fig. 2(b)). By selecting the "Program list" icon a menu is displayed (Fig. 3(b)) that allows the user to select and play video content in the same way. With an open hand posture (Fig. 2(c)), the user can return to the main menu.

3. Hand detection using AdaBoost

The object detection method using joint rectangle features [4] is used for detecting hand regions. This is an extension of Viola and Jones' method [7] where higher classification performance can be achieved by evaluating co-occurrence of multiple rectangle features in each weak classifier.

One detector for each hand posture is scanned across the image at multiple scales as shown in Figure 4, where each detector consists of a cascade of strong classifiers. The initial strong classifier rejects the majority of sub-windows with very little processing time. Sub-windows that are not rejected by the initial classifier are processed by subsequent strong classifiers, each more complex than the previous one. If any strong classifier rejects the sub-window, no further processing is performed.

Figure 5 shows an example of strong classifiers. The output H of a strong classifier is computed as a linear combination of T weak classifiers h_t:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right), \quad (1)$$

where α_t is the weight of weak classifier t determined through the learning process using example hand and non-hand images.

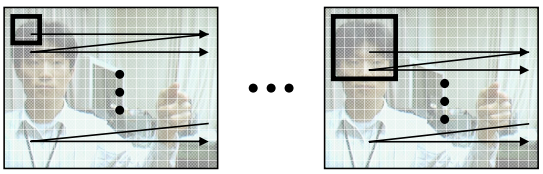


Figure 4. Multi-scale detection by evaluating sub-windows of N different sizes.

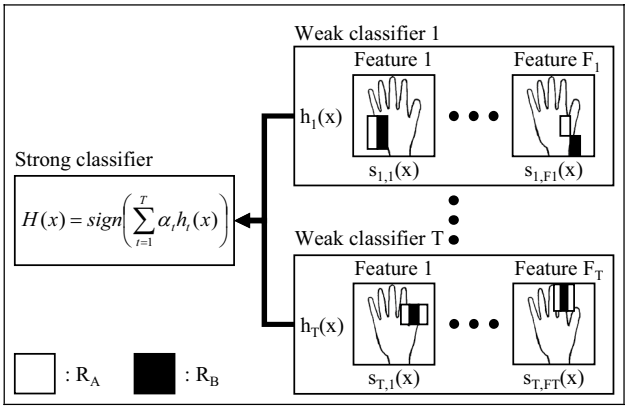


Figure 5. Combining weak classifiers into a strong classifier. Note that in contrast to Viola and Jones' algorithm, each weak classifier can use multiple rectangle features.

Each weak classifier makes use of multiple rectangle features. This combined feature is called joint rectangle feature. Each rectangle feature has a scalar value z that represents differences in average intensities between two rectangular regions; R_A and R_B. The value z can be efficiently calculated from the pre-computed integral image [7]. The variable s is calculated by

$$s(x) = \begin{cases} 1 & \text{if } p \cdot z(x) > p \cdot \theta \\ 0 & \text{otherwise} \end{cases}, \quad (2)$$

where θ is a threshold and p is a parity indicating the direction of the inequality sign.

The joint rectangle features are represented by combining the binary variables computed from multiple features. The value of joint rectangle feature j is calculated by

$$j_t(x) = \sum_{i=1}^{Ft} 2^{i-1} \cdot s_{t,i}(x), \quad (3)$$

where F_t is the number of combined features in weak classifier t and $s_{t,i}(x)$ is the variable obtained by quantizing $z_{t,i}(x)$, which represents whether it is a hand image or not.

The value of each weak classifier h is determined by the feature value of the joint rectangle feature j as

$$h_i(x) = \begin{cases} +1 & \text{if } P(y=+1 \mid j) > P(y=-1 \mid j) \\ -1 & \text{otherwise} \end{cases}, \quad (4)$$

where $P(y=+1 \mid j)$ and $P(y=-1 \mid j)$ are joint probabilities observing feature co-occurrence represented by j. The values of these joint probabilities are also determined during the learning process.

4. Optimizing detection for multi-core processors

To achieve the best performance on multi-core processors we need to optimize the operation by equally

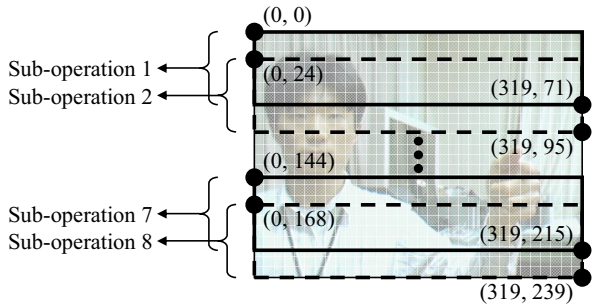
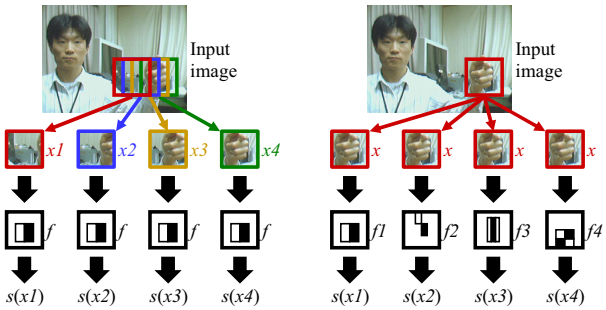


Figure 6. Sub-dividing the operation on an input frame and distributing the sub-windows to each processing core.



(a) Window-parallel (b) Feature-parallel

Figure 7. Two types of parallel operations are used for speeding up the feature computation using SIMD operations.

sub-dividing and distributing it to the processing cores. It is also important to minimize the communication between the modules because it takes a certain time to begin communications. Using exclusive local memory associated with each core helps to minimize the communication by allocating all data required for the operation. In the following section, we describe how we sub-divide the detection algorithm for multi-core processors.

4.1. Sub-dividing the operation

In the cascaded detection method the operations for each detection sub-window can be executed in parallel, see Fig. 6. When dividing the target image into several sub-images each sub-image should contain all pixels scanned by the corresponding detection window. Therefore, there are pixels in the target image that belong to more than one sub-images. The height of the shared part is $w - d$, where w and d are the detection window size and the step size, respectively. Consequently, the height of each sub-image h_{SUB} required for these sub-operations is given by

$$h_{SUB} = (w-d) + \frac{h_{IMG} - (w-d)}{n}, \qquad (5)$$

where h_{IMG} and n are the height of the target image and the number of sub-operations, respectively.

Fig. 6 shows the result of dividing the 320x240 target image into eight sub-images for detection by a 50x50 detection sub-window and step size 2. If each processing core has exclusive local memory, transmitting each sub-image to local storage significantly increases the detection speed.

When sub-dividing the operation, we must take care not to divide into too many operations. The height of each sub-image is 72 pixels in the case of Fig. 6. The sum of the height of all sub-images is 576 pixels, 2.4 times as large as the target image. This increases the time for transmitting these sub-images to processing cores. The optimal number of divisions depends on the size of the target image, the size of the detection sub-window, the communication speed between modules and the number of processing cores. If the communication speed is fast enough, it is optimal to set the number of divisions n to the smallest multiple of the number of cores where the size of each sub-image is smaller than the size of the local storage.

4.2. Hand detection in large image regions

In multi-core processors, the size of local storage is generally small because of hardware restrictions. This limits the maximum size of the detection window. To overcome this restriction we use an image pyramid. First, two down-sampled images of sizes 160x120 and 80x60 pixels are created from the 320x240 pixel input image. This limits the required maximum size of the detection window and allows the detection of hand poses close to the camera. For example, when searching for hand images at a scale of 80x80 pixels, we can use a 160x120 down-sampled image with a 40x40 detection window.

When using down-sampled images, one should take care not to use too small detection windows. Experiments confirmed that the false positive rate increases when the detection window is smaller than in the original detection cascade. For example, if the size of the original detection window is 25x25 pixels, the down-sampled image should only be used for detecting hand regions larger than 50x50 pixels.

4.3. Parallel feature computation

In addition to distributing the operation to multiple cores we introduce two types of parallel operations on each core to accelerate the feature computation: The first method is to use window-parallel operations. As shown in Fig. 7(a), the feature values $s(x1)$, $s(x2)$, $s(x3)$ and $s(x4)$ are calculated for four neighboring detection sub-windows $x1$, $x2$, $x3$ and $x4$ in parallel by 4-way SIMD operations, resulting in a speed-up factor of four. However, the performance depends on the input image because the feature calculation cannot be terminated until all detection windows have been rejected by the cascade, while the calculation can be terminated when one of the strong classifiers rejects the current detection window in a single feature calculation.

If just one of four detection sub-windows remains as a candidate, there is no advantage of window-parallel operations. In this case the system focuses on this sub-window by using feature-parallel operations. As shown in Fig. 7(b), the feature values $s1$, $s2$, $s3$ and $s4$ of four features $f1$, $f2$, $f3$ and $f4$ are calculated for a single sub-window x in parallel by 4-way SIMD operations. Although the performance gain of feature-parallel operations on average is slightly smaller than that of window-parallel operations (because SIMD operations are not available for loading feature information) the operating speed is still faster than without parallel operations. The effective performance of the weak classification operation depends on the number of component features, but can be up to four times faster than the weak classification operation without

Table 1. Operating speed for detecting hand postures in a single image

	Cell Reference Set (CBE 2.8GHz)	Single-core system (Xeon 2.8GHz)
Operating speed (average)	24~44 [ms] (34 [ms])	263~356 [ms] (325 [ms])
Ratio	7.2~13.5	1

Table 2. Average false detection (false positives) and missed detection (false negatives) rates are shown for each hand posture computed from 6000 images from various users and backgrounds.

	Pointing hand	Fist	Open hand
False pos rate	4.6%	0.1%	3.5%
False neg rate	0.0%	4.6%	0.9%

feature-parallel operations.

5. Evaluation

This section shows experimental results of evaluating the system performance and usability. To confirm the effectiveness of the proposed method, we have implemented the hand gesture recognition on the prototype version of the Cell Broadband Engine™ (CBE). It has eight cores which are called Synergistic Processor Elements (SPEs) operating at 2.8GHz[1]. Each SPE can execute 4-way 32-bit SIMD operations independent of other SPEs and has 256KB exclusive local storage for minimizing the data communication between modules. Using our optimized method, the detection can be executed much faster than on the single processor system.

5.1. Computation time

We evaluated the operating speed of hand gesture recognition on our system compared with the same operation on a PC with a single-core Xeon processor. The operating speed of hand gesture recognition depends on the input image and the classification cascade used for recognizing hand gestures. We compare the two systems under the same conditions using a test sequence of 160 frames. To compare the performance between a multi-core system and a single-core system, we disabled the hyper-threading function of a Xeon processor.

The result is shown in Table 1. We measured the average, the best case, and the worst case of the operating speed for recognizing the three hand postures shown in Figure 2. The average detection time on the PC is 325ms, while the average time on the CRS is 34ms. The operation on the CRS is fast enough to realize interactive gesture recognition.

The CBE can execute up to 32 32-bit operations in parallel, while a single-core Xeon processor can execute up to 4 integer additions and subtractions, or up to 2 operations in each clock cycle. Thus the expected operating speed of our system is 8 to 16 times as fast as on a PC. The result shows that the operating speed on the reference set is up to 13.5 times faster than on a PC.

[1] The release version of CBE has 7 SPEs operating at 3.2GHz.

5.2. Reliability

The reliability of hand gesture detection is evaluated by measuring the error rate for each hand posture. Table 2 shows the error rate for each hand posture. The error rates are calculated from the number of erroneous detections in approximately 6000 images including seven users in front of different backgrounds.

Note that the error rate is affected by the background scene. For example, when an object with many vertical edges is in the background, the error rates for the open hand increases because these hand postures also have a lot of vertical edges. However, the error rate is sufficiently low for controlling consumer electronics under ordinary conditions because the joint rectangle features, capturing local intensity gradients and orientations, are sufficiently discriminative.

6. Conclusion

For real-time hand gesture recognition we have optimized the hand posture detection for multi-core processors. The operations are divided and distributed among the processing cores and processed using two kinds of parallel operations. Implementing the algorithm on the Cell Broadband Engine allows us to realize much faster operation compared with a single-core processor. This contributes to making the hand gesture UI more responsive and thus more user-friendly.

Acknowledgments

We wish to thank Takeshi Mita and Yasuhiro Taniguchi of Toshiba Corp. for the contribution in developing the system. We also thank Toshiba Semiconductor Company for the support of developing the system using the CRS.

References

[1] L. Bretzner, I. Laptev, T. Lindeberg, “Hand Gesture Recognition using Multi-Scale Colour Features, Hierarchical Models and Particle Filtering”, Proc. Int. Conf. on Face and Gesture Recognition, pages 405-410, 2002.

[2] W. T. Freeman, C. D. Weissman, “Television control by hand gestures”, Proc. Int. Conf. on Face and Gesture Recognition, 1995.

[3] M. Kölsch, M. Turk, “Robust Hand Detection”, Proc. Intl. Conf. Autom. Face and Gesture Recognition, p. 614-619, 2004.

[4] T. Mita, T. Kaneko, O. Hori, “Joint Haar-like Features for Face Detection”, Proc. ICCV, pp.1619-1626, 2005.

[5] E.-J. Ong, R. Bowden, “A Boosted Classifier Tree for Hand Shape Detection”, Intl. Conf. Autom. Face and Gesture Recognition, p. 889-894, 2004.

[6] B. Stenger, “Template-Based Hand Pose Recognition Using Multiple Cues”, Proc. ACCV, p. 551-560, 2006.

[7] P. Viola, M. Jones, “Rapid Object Detection using a Boosted Cascade of Simple Features”, Proc. CVPR, p.511-518, 2001.