An FPGA-based Parallel Hardware Architecture for Real-time Eye Detection

Dongkyun Kim*, Junhee Jung*, Thuy Tuong Nguyen*, Daijin Kim**, Munsang Kim***, Key Ho Kwon*, and Jae Wook Jeon*

Abstract—Eye detection is widely used in applications, such as face recognition, driver behavior analysis, and human-computer interaction. However, it is difficult to achieve real-time performance with software-based eye detection in an embedded environment. In this paper, we propose a parallel hardware architecture for real-time eye detection. We use the AdaBoost algorithm with modified census transform (MCT) to detect eyes on a face image. We parallelize part of the algorithm to speed up processing. Several downscaled pyramid images of the eye candidate region are generated in parallel using the input face image. We can detect the left and the right eye simultaneously using these downscaled images. The sequential data processing bottleneck caused by repetitive operation is removed by employing a pipelined parallel architecture. The proposed architecture is designed using Verilog HDL and implemented on a Virtex-5 FPGA for prototyping and evaluation. The proposed system can detect eyes within 0.15 ms in a VGA image.

Index Terms—Eye detection, hardware architecture, FPGA, image processing, HDL

I. INTRODUCTION

Eye detection is an image processing technique to locate eye position from an input facial image. Eye detection is widely used in applications, including face recognition, driver behavior analysis, and human-computer interaction [1-4]. The extraction of local facial features, such as nose, eye, and mouth, is important in face recognition to analyze the face image [1-2]. The eye is the most important and most commonly adopted feature for human recognition [2]. Eye detection is used to track the eye and gaze to monitor driver vigilance in driver behavior analysis [3]. Eye-gaze tracking system can be used for pointing and selection in human-computer interactions [4].

Z. Zhu and Q. Ji classified traditional eye detection methods into three categories: template based methods, appearance based methods, and feature based methods [5]. In the template-based methods, a generic eye model is first designed and the template matching is used to detect the eyes on the image. The deformable template method is a commonly used template based method [6-8]. In this method, an eye model is first designed and then the eye position is obtained via a recursive process. This method can detect the eye accurately, but it is computationally expensive and good image contrast is required for the method to converge correctly.

The appearance-based methods detect eyes based on their photometric appearance [9-11]. These methods usually need a large amount of training data representing the eyes of different subjects, under different face orientations, and under different illumination conditions. These data are used to train a classifier; detection is then achieved via classification. The feature-based methods use characteristics, such as the edge, iris intensity, color distribution, and the flesh of the eyes to identify features...
around the eyes [12].

There are several approaches to reduce eye detection processing time. These approaches can be broadly classified into two groups: software-based and hardware-based. The focus in the software-based approach is designing and optimizing an eye detection algorithm for real-time performance. Conversely, in the hardware-based approach, the eye detection algorithm is parallelized and implemented on DSP or FPGA to achieve real-time performance.

T. D’Orazio et al. propose a real time eye detection algorithm [13]. They use iris geometrical information to determine the eye region candidate and symmetry to select the pair of eyes. Their algorithm consists of two steps. First, the region candidate that contains one eye is detected from the whole image by matching the edge directions with an edge template of the iris. Then, in the opposite region, a search is carried out for the second eye, whose distance and orientation are compatible with the range of possible eye positions. The software has been implemented using Visual C++ on a Pentium III 1GHz. The search for the eyes in an image of 640×480 takes about 0.3 sec.

K. Lin et al. proposed a fast eye detection scheme for use in video streams rather than still images [14]. The temporal coherence of sequential frames was used to improve the detection speed. First, the eye detector trained by the AdaBoost algorithm is used to roughly obtain the eye positions. Then, these candidate positions are filtered according to geometrical patterns of human eyes. The detected eye regions are then taken as the initial detecting window. The detecting window is updated after each frame is detected. The experimental system was developed with Visual C++ 6.0 on a Pentium 2.8 GHz and 1 GB memory. The mean detection rate was 92.73% for 320×240 resolution test videos in their experiments, with a speed of 24.98 ms per frame.

However, CPU-based implementation has two major bottlenecks that limit performance: data transfer bandwidth and sequential data processing rate [15]. First, video frames are captured from the camera and transferred to the CPU main memory via a transfer channel, such as USB or IEEE1394. These channels have limited bandwidth; hence they impose a data flow bottleneck. After a frame is grabbed and moved to memory, the CPU can sequentially process the pixels. The CPU adds a large overhead to the actual computation. Its time is split between moving data between memory and registers, applying arithmetic and logic operators on the data, and maintaining the algorithm flow, handling branches, and fetching code. Additionally, a relatively long latency is imposed by the need to wait for an entire frame to be captured before it is processed.

The Deployment and real-time processing of the eye detection algorithm in an embedded environment are difficult due to these reasons. Therefore, A. Amir et al. implemented an embedded system for eye detection using a CMOS sensor and FPGA to overcome the shortcomings of CPU-based eye detection [15]. Their hardware-based embedded system for eye detection was implemented using simple logic gates, with no CPU and no addressable frame buffers. The image-processing algorithm was redesigned to enable highly parallel, single-pass image-processing implementation. Their system processes 640×480 progressive scan frames at a rate of 60 fps, and outputs a compact list of eye coordinates via USB communication. It is difficult to achieve real-time performance with CPU-based eye detection in an embedded environment. A. Amir et al. reduce processing time by using FPGA [15]. However, the eye detection algorithm they used is relatively simple; hence, the eye miss rate is relatively high.

B. Jun and D. Kim proposed a face detection algorithm using MCT and AdaBoost [16]. I. Choi and D. Kim presented an eye detection method using AdaBoost training with MCT-based eye features [17]. The MCT-based AdaBoost algorithm is very robust, even when the illumination varies, and is very fast. They choose the MCT-based AdaBoost training method to detect the face and the eye due to its simplicity of learning and high speed of face detection.

In this paper, we propose a dedicated hardware architecture for eye detection. We use the MCT-based AdaBoost eye detection algorithm proposed by I. Choi and D. Kim [17]. We revise the algorithm slightly for parallelization. We parallelize the part of the algorithm that is iterated to speed up processing. Several downsampled images of eye candidate regions are generated in parallel using the input face image in the pyramid downscaling module. These images are passed to the MCT and cascade classifier module via a pipeline
and processed in parallel. Thus, we can detect the left and right eye simultaneously. The sequential data processing bottleneck caused by repetitive feature evaluation is removed by employing pipelined parallel architecture.

The proposed architecture is described using Verilog HDL (Hardware Description Language) and implemented on an FPGA. Our system uses CameraLink to interface with the camera. The CameraLink interface transfers the control signal and data signal synchronously. The entire system is synchronized with a pixel clock signal from the camera. Our system does not need an interface such as USB; hence we have removed the bottleneck caused by the interface bandwidth. The frame rate of the proposed system can be flexibly changed in proportion to the frequency of the camera clock. The proposed system can detect left and right eyes within 0.15 ms using a VGA (640×480) image and a 25 MHz clock frequency.

This paper consists of six sections and appendices. The remainder of the paper is organized as follows. In Section 2, we describe the details of the MCT and AdaBoost algorithm and the overall flow of the eye detection algorithm that we use. Section 3 describes the proposed hardware architecture and the simulation results. Section 4 presents the configuration of the system and the implementation result. Section 5 presents the experimental results. Finally, our conclusions are presented in Section 6.

**II. EYE DETECTION ALGORITHM**

1. Modified Census Transform

The modified census transform is a non-parametric local transform that is a modification of the census transform by B. Fröba, and A. Ernst [18]. It is an ordered set of comparisons of pixel intensities in a local neighborhood to find out which pixels have lower intensity than the mean pixel intensity. In general, the size of the local neighborhood is not restricted, but in this work, we always assume a 3×3 surrounding.

The modified census transform is described below. We define N(x) as the local spatial neighborhood of the pixel x, so that x \∉ N(x), and N'(x) = N(x) ∪ x. Let I(x) and \( \overline{T}(x) \) be the intensity of the pixel x and the intensity mean in the neighborhood N'(x), respectively. The modified census transform can be presented as

\[
\Gamma(x) = \bigotimes_{x \in N'(x)} \xi(\overline{T}(x), I(x'))
\]

where \( \xi(\overline{T}(x), I(x')) \) is a comparison function which yields 1 if \( \overline{T}(x) < I(x') \), and yields 0 if \( \overline{T}(x) \geq I(x') \), and \( \bigotimes \) is a concatenation operation to form a bit vector (see Fig. 1). If we consider a 3×3 neighborhood, we are able to determine 511 distinct structure kernels. Fig. 1 shows an example of the modified census transform.

In this example, I(x) is 95.11. Each pixel on N'(x) is transformed to 0 or 1 by applying \( \xi(\overline{T}(x), I(x')) \). Bits are concatenated in order, as shown as Fig. 1.

2. AdaBoost Algorithm

AdaBoost, short for Adaptive Boosting, is a machine learning algorithm, formulated by Y. Freund and R. Schapire [19]. They introduced the AdaBoost algorithm briefly in another work and showed that it had good generalization capability [20]. It is a meta-algorithm, and can be used in conjunction with many other learning algorithms to improve the performance. AdaBoost is adaptive, in the sense that subsequent classifiers that are built are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. However, in some situation it can be less susceptible to the over fitting problem than most learning algorithms [21].

AdaBoost calls a weak classifier repeatedly in a series of rounds \( t = 1, 2, \ldots, T \) from a total of \( T \) classifiers. For each call, a distribution of weights \( DT \) is updated that indicates the importance of examples in the data set for the classification. The weights of each incorrectly classified example are increased (or alternatively, the
weights of each correctly classified example are decreased), in each round, so that the new classifier focuses more on those examples [21].

3. MCT-based AdaBoost Algorithm

In this paper, we use the AdaBoost training method with MCT-based eye features proposed by I. Choi and D. Kim [17]. In the AdaBoost training, they constructed a weak classifier that classifies the eye and non-eye patterns and then constructed a strong classifier that is a linear combination of weak classifiers [17].

In the detection, they scanned the eye candidate region by moving a 12×8 sized part of the scanning window and obtained a confidence value corresponding to the current window location using the strong classifier. Then, they determined the window location whose confidence value was maximized as the location of the detected eye. Their MCT-based AdaBoost training was performed using only the left eye and non-eye training images. Therefore, when they tried to detect the right eye, they needed to flip the right subregion of the face image [17].

Fig. 2 shows the flowchart of the eye detection algorithm. The eye detection algorithm requires a face image and the coordinates of the face region as inputs. If the detection completes successfully, the algorithm outputs the coordinates of the left and right eyes as results.

Eye candidate regions are cropped for pyramid downscaling in the first step. The face region is split in half horizontally and vertically. The top-left subregion is set as the left eye candidate region and top-right subregion is set as the right eye candidate region. Then, the right eye candidate region is flipped, because the training data for the right eye is the same as for the left. These candidate images are down-scaled to a specific size to get pyramid images. There are eight pyramid images (four for the left and four for the right) and the images sizes are 33×36, 31×34, 29×32, and 27×29. The four down-scaled images are enough for an indoor environment where the support distance is 0.5 ~ 8 meters. These sizes of the down-scaled images were determined by empirical experiments.

MCT is applied to the downscaled pyramid images in the second step. The window size for the transform is 3×3. We can get MCT images by sliding the window across the pyramid image. Then, the MCT-transformed pyramid images are passed to the cascade classifier to detect eyes.

The MCT images are scanned by the classifier and candidate eye locations are extracted in the third step. There are three stages of classifiers. The window size for the classifier is 15×15. The classifier slides the window across the MCT image and extracts the confidence value of the current window position. If the confidence value is lower than the specific threshold value, then the confidence value of the next-stage classifier is calculated. If the confidence value of the last stage is lower than the threshold, then an eye candidate is detected in the current window position. The eye coordinate can be calculated from the current window position. If there are no detected eye candidates after sliding, then detection has failed.

Eye detection for one pyramid image is completed in the third step. The algorithm repeats these steps for all the pyramid images. After the left eye detection finishes, the same process is applied to the right eye pyramid images. After this repetitive operation, the left eye candidates list and right eye candidates list are updated. In the post-processing step, if there are several eye candidates in the list, the eye candidate that has the maximum confidence value is selected as the result.

![Fig. 2. Eye detection algorithm flowchart.](image-url)
III. Proposed Hardware Architecture

1. Overall Architecture

Fig. 3 shows the overall hardware architecture of the proposed system. The proposed system consists of four sub-modules: a pyramid downscaling module, an MCT module, a parallel classifier module, and a post-processing module. The face detector and SRAM are not part of the proposed system.

The face detector receives the image from the camera and detects faces on the image. The face detector outputs the top-left and bottom-right coordinates of the face region. The face detector passes the face region coordinates to the eye detection module after detecting the faces. The SRAM Interface module updates every frame of the image from the camera to SRAM. This image is a non-uniform eye region image. A stored non-uniform eye image is then used to generate the fixed size pyramid image by using the pyramid downscaling module.

The pyramid downscaling module receives as input the left and the right eye region images from SRAM. SRAM cannot be accessed in parallel, so we have to crop the image in serial. First, the downscaled images of the left and right eye regions are generated using SRAM data and stored in registers. After generating the first downscaled images, we can generate other downscaled images in parallel, because the registers can be accessed in parallel. This is different to a software algorithm, in which pyramid images are generated one-by-one.

All generated pyramid images are passed to the MCT module in parallel through pipelines. There are eight MCT modules for eight pyramid images. In the MCT modules, the $3 \times 3$ window slides across the pyramid images for the census transform. MCT images are passed to the classifier module in parallel through pipelines.

The classifier module extracts features from the MCT image. In this module, a window with a pre-defined size is used to scan through the MCT image from top-left to bottom-right. At every stage of the window scanning, if the confidence value of this stage is greater than a defined threshold, then the current window region is considered as an eye candidate. The threshold values are built in advance (by learning), corresponding to the classifier’s stages. The coordinates of the eye candidate can be calculated based on the current position of the scanning window. In contrast to the software implementation, in this paper we apply the 3-stage classifier in parallel for eight MCT images. Hence, we need 24 classifiers (3 stages $\times$ 8 images) for detection. Moreover, our hardware implementation can detect the left eye and the right one simultaneously. As a result, several eye candidates can be detected within one left or right eye region. If there are more than one eye candidate, the candidate that has the maximum confidence value is selected as the final detection result.

2. Flip / Pyramid Downscaling Module

Fig. 4 shows the pyramid downscaling images of eye candidate regions. The pyramid downscaling module needs a face image and face region coordinates to generate pyramid images. The face image is stored in SRAM and face region coordinates can be received from the face detector.

First, the pyramid module splits the eye candidate regions. As shown in Fig. 4, the top-left region is set as the left eye candidate region and the top-right region is set as the right eye candidate region. Image pixels in SRAM can be accessed one-by-one every pixel clock period. Thus, we cannot generate all eight pyramid images simultaneously. Therefore, we should generate the first pyramid images of the left and right eye regions
in serial. In Fig. 4, image (1) and image (2) are generated in serial, and these image data are stored in registers. We use nearest-neighbor interpolation to generate downscaling images. We can generate other downscaled images in parallel using these images stored in registers.

In the software algorithm, the right eye is flipped after cropping the image. As shown in Fig. 4, the right eye candidate region is scanned from right to left. Thus, flip is unnecessary in our architecture. The size of pyramid images is the same as in the software algorithm (33×36, 31×34, 29×32, 27×29).

3. MCT Module

Fig. 5 shows the realization of MCT bit computation. The average intensity of pixels in the 3×3 window is needed to calculate the MCT bit vector. We need to use the divider module to calculate the average intensity. However, the computation latency of the divider module is very long compared to the adder module. Thus, we use the adder and shifter instead of the divider.

We do not calculate average intensity. Instead, we calculate SUM_1 and I(x,y)×9 as shown in Fig. 5. SUM_1 is the sum of pixels in the window. We can calculate this value using the adder tree. The I(x,y)×9 value can be calculated by multiplying the intensity of every pixel by nine. We implement the computation using a combination of the shifter and adder, as shown in Figure 8. By comparing SUM_1 and I(x,y)×9, we can get the number of MCT bits for each pixel in the window.

4. Parallel Classifier Module

In the classifier module, a window of a pre-defined sized (15×15) slides from top-left to bottom-right on the MCT images, as shown in Fig. 6. In contrast to the software algorithm, we apply the 3-stage classifier in parallel for eight MCT images. Thus, we can calculate confidence values for each stage classifier simultaneously. We can detect both eyes simultaneously. We need 24 classifiers (8 images × 3 stages).

We use a revised version of the parallel classifier cascade architecture proposed by Jin et al. [22] for eye detection, as shown in Fig. 7. They designed the

---

**Fig. 4.** Pyramid downscaling images.

**Fig. 5.** Realization of MCT bit computation.

**Fig. 6.** Parallel classifier cascade for left eye detection.
classifiers using look-up tables (LUTs), since each classifier consists of the set of feature locations and the confidence value corresponding to the LBP (local binary pattern). LUTs contain training data generated by AdaBoost training. We revise this LUT for MCT rather than LBP. The three stages of the cascades have an optimized number of allowed positions: 22, 36, and 100, respectively. In the software algorithm, the classifiers are cascaded in serial. In the proposed hardware, however, all the cascade stages are pipelined and operate in parallel. The feature extraction result is generated at each pixel clock time after the fixed pipeline latency.

IV. SYSTEM IMPLEMENTATION

1. Synthesis Result

All hardware modules are implemented using Verilog HDL and simulated by the ModelSim 6.5 Simulator for functional verification. We use a camera with a 24.5454 MHz clock frequency for the implementation, so for consistency we use a 25 MHz clock frequency for the simulation.

The hardware module is synthesized by the Synplify logic synthesis tool from Synplicity. The target device for the implementation is a Virtex-5 LX330 FPGA from Xilinx. We use the Xilinx ISE 10.1.3 integration tool for front-end and back-end implementation. The place and route phase is performed by Xilinx PAR (Place and Route Program).

Table 1 shows the device utilization summary reported by the Xilinx ISE. The number of occupied slices and the total used memory of the proposed system are 39,914 (about 76% of the device) and 288 KB, as shown in Table 1. Its equivalent gate count is about 200 K gates.

Table 2 shows the timing summary reported by the Synplify synthesis tool. The requested frequency is 25.0 MHz and the maximum estimated frequency of the implemented system is 106.3 MHz. We use a progressive camera that has a 24.5454 MHz pixel clock. This camera captures the VGA image at 60 fps. We can expect a rate of around 240 fps for the processing of standard VGA images at the reported maximum estimated frequency of the system, because the frame rate increases linearly with the pixel clock increment.

2. System Configuration

Fig. 8 shows the system configuration for testing and verifying the eye detection system. The overall system consists of four PCBs (Printed Circuit Board): a CameraLink interface board, two FPGA boards for face and eye detection, and an SRAM/DVI interface board.

The CameraLink interface board receives sync signals and image data signals from the camera using the CameraLink interface. These signals include pixel clock, VSYNC (vertical sync - frame valid), HSYNC (horizontal sync - line valid), DVAL (data valid) and streaming image data. The FPGA boards grab these signals for use...
in face and eye detection. We use two FPGA boards. One is used for face detection and the other is used for eye detection. We use the face detector proposed by Jin et al. [22] to detect the coordinates of face regions. The single image frame is stored and updated in SRAM to generate pyramid downscaling images. The eye detection result is displayed on the monitor using a DVI interface.

V. EXPERIMENTAL RESULT

The robustness of our system was evaluated using both the provided image database and real people’s faces captured by our camera system. Images from the facial database were downloaded and referred from [23]. We selected the first directory named 2002-07 in the database for the experiment. In this directory, there are 934 images containing 1394 faces or left-right eye pairs. The evaluation was made using four criteria. First, the precision and recall values were analyzed with respect to all left-right eye pairs. These values were calculated based on the number of correct detections (true positives), the number of missed detection (false negatives), and the number of wrong detections (false positives). Second, the result was analyzed when out-of-focus images were not considered. Similarly, in the two remaining criteria, rotation-in-plane (RIP) and with-glasses were not considered, respectively. Fig. 9 shows the experimental results in four different cases. It is noted that our implementation does not support the case of rotation-out-of-plane (ROP); thus, all faces rotated out of the plane (i.e. a face that contains only one eye) were not considered in the result evaluation. The experimental results yielded 93% detection accuracy. Fig. 10 shows sample image results where faces and eyes were successfully detected under different conditions.

Fig. 11 and Table 3 show the detection results of real people’s faces captured by our camera system, with and without glasses, respectively. The detection rate is 87% from 1200 frames. As shown in Table 3, the detection rate of people wearing glasses drops significantly. There can be light spots or blooming near eyes due to light reflection caused by glasses. These effects can influence the detection rate. Generally, the detection rate of people with glasses is lower than without glasses. Therefore, we need some additional processing to reduce the effects caused by glasses.

By considering the detection results for the provided image database (software implementation) and the real
people’s faces captured by our camera system (hardware implementation), we can see that the detection rate of the hardware implementation is lower than that of the software implementation in the case of with-glasses. However, the processing time of the hardware system is much better than that of the software: 0.15 ms/frame versus 20 ms/frame. We used a personal computer for the evaluation of the software implementation, which had an AMD Athlon 64 X2 Dual Core Processor 3800+ 2.00 GHz and 2-GB RAM.

Table 4 shows a comparison chart with other eye detection algorithms. Our method has the largest image size and fastest processing speed, even though small decrease of detection accuracy. Our method is suitable for hardware implementation.

Table 3. Detection results for persons with and without glasses

<table>
<thead>
<tr>
<th></th>
<th>Glasses</th>
<th>Frames</th>
<th>Falsey Detected</th>
<th>Detection Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person 1</td>
<td>Yes</td>
<td>300</td>
<td>43</td>
<td>85.67 %</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>300</td>
<td>10</td>
<td>96.67 %</td>
</tr>
<tr>
<td>Person 2</td>
<td>Yes</td>
<td>300</td>
<td>85</td>
<td>71.67 %</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>300</td>
<td>17</td>
<td>94.33 %</td>
</tr>
<tr>
<td>Total</td>
<td>-</td>
<td>1200</td>
<td>155</td>
<td>87.08 %</td>
</tr>
</tbody>
</table>

Table 4. Performance summary of reported software-based eye detection systems

<table>
<thead>
<tr>
<th>Eye Detection Software Implementations</th>
<th>Image Size</th>
<th>Detection Rate</th>
<th>FPS</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>K. Peng 2005 [24]</td>
<td>92 x 112</td>
<td>95.2 %</td>
<td>1   fps</td>
<td>Feature Localization, Template Matching</td>
</tr>
<tr>
<td>Q. Wang 2006 [26]</td>
<td>384 x 286</td>
<td>95.6 %</td>
<td>-</td>
<td>Binary Template Matching, Support Vector Machines</td>
</tr>
<tr>
<td>T. Akashi 2007 [27]</td>
<td>320 x 240</td>
<td>97.9 %</td>
<td>35  fps</td>
<td>Template Matching, Genetic Algorithm</td>
</tr>
<tr>
<td>H.-J. Kim 2008 [28]</td>
<td>352 x 240</td>
<td>94.6 %</td>
<td>5   fps</td>
<td>Zernike moments, Support Vector Machines</td>
</tr>
<tr>
<td>M. Shafi 2009 [29]</td>
<td>480 x 540</td>
<td>87 %</td>
<td>1   fps</td>
<td>Geometry, shape, density features</td>
</tr>
<tr>
<td>Ours</td>
<td>640 x 480</td>
<td>93 %</td>
<td>50  fps</td>
<td>AdaBoost, Modified Census Transform, Multi-scale Processing</td>
</tr>
</tbody>
</table>

VI. CONCLUSIONS

In this paper, we proposed a dedicated hardware architecture for eye detection. We implemented it on an FPGA. The implemented system processes each pyramid image of the left and the right eye candidate regions in parallel. The processing time to detect the left and right eyes of one face is less than 0.15 ms with a VGA image. The maximum estimated frequency of the implemented system is 106.3 MHz. We can expect a rate of around 240 fps for the processing of standard VGA images at the reported maximum estimated frequency of the system, because the frame rate increases linearly with the pixel clock increment.

The detection rate of our system is 93% with the FDDB face database and 87% for real people’s faces. However, the detection rate for people with glasses drops significantly. This is a weakness of our proposed system. In the future, we will add a pre-processing step to reduce the effects caused by glasses.

ACKNOWLEDGMENTS

This work was supported by the Ministry of Knowledge Economy and the Korea Institute for Advancement in Technology through the Workforce Development Program in Strategic Technology.

REFERENCES


Dongkyun Kim received B.S. and M.S. degrees in electrical and computer engineering from Sungkyunkwan University, Suwon, Korea, in 2007 and 2009, respectively. He is currently working toward a Ph.D. degree with the School of Information and Communication Engineering, Sungkyunkwan University. His research interests include image/speech signal processing, embedded systems, and real-time applications.

Junhee Jung received a B.S. degree in the School of Information and Communication Engineering from Sungkyunkwan University, Korea, in 2009 and an M.S. degree in the Department of Electrical and Computer Engineering from Sungkyunkwan University, Korea, in 2011, respectively. His interests include image processing, computer vision and SoC.

Thuy Tuong Nguyen received a B.S. (magna cum laude) degree in computer science from Nong Lam University, Ho Chi Minh City, Vietnam, in 2006, M.S. and Ph.D. degrees in electrical and computer engineering from Sungkyunkwan University, Suwon, Korea, in 2009 and 2012, respectively. His research interests include computer vision, image processing, and graphics processing unit computing.

Daijin Kim received a B.S. degree in electronic engineering from Yonsei University, Seoul, Korea, in 1981, an M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Taejon, 1984, and a Ph.D. degree in electrical and computer engineering from Syracuse University, Syracuse, NY, in 1991. He is currently Professor in the Department of Computer Science Engineering at POSTECH. His research interests include intelligent fuzzy system, soft computing, genetic algorithms, and hybridization of evolution and learning.
Munsang Kim received his B.S. and M.S. degrees in mechanical engineering from the Seoul National University in 1980 and 1982, respectively, and the Dr.-Ing. degree in robotics from the Technical University of Berlin, Germany, in 1987. Since 1987, he has been working as a research scientist at the Korea Institute of Science in Korea. He leads the Advanced Robotics Research Center since 2000 and became the director of the Intelligent Robot-The Frontier 21 Program since 2003. His current research interests are the design and control of novel mobile manipulation systems, haptic device design and control, and sensor application in intelligent robots.

Key Ho Kwon received a B.S. degree from the Department of Electronics Engineering, Seoul National University, Korea, in 1975, an M.S. degree from the Department of Electronics Engineering, Seoul National University, Korea, in 1978, and a Ph.D. degree from the Department of Electronics Engineering, Seoul National University, Korea, in 1989. He is currently a Professor at the School of Information and Communication Engineering, Sungkyunkwan University. His research interests include artificial intelligence, fuzzy theory, neural networks, and genetic evolutionary algorithms.

Jae Wook Jeon received B.S. and M.S. degrees in electronics engineering from Seoul National University, Seoul, Korea, in 1984 and 1986, respectively, and a Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, in 1990. From 1990 to 1994, he was a Senior Researcher with Samsung Electronics, Suwon, Korea. Since 1994, he has been with the School of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, first as an Assistant Professor, and currently as a Professor. His research interests include robotics, embedded systems, and factory automation.