Lines that will appear in the proceedings

Lines that will appear in the program

Implementing Image Processing Algorithms on FPGA-based Realtime Vision System

Shinichi Hirai Masakazu Zakouji Tatsuhiko Tsuboi

Department of Robotics, Ritsumeikan University
Kusatsu, Shiga 525-8577, Japan

hirai@se.ritsumei.ac.jp sme40109@se.ritsumei.ac.jp rr004970@se.ritsumei.ac.jp

Abstract— In this article, we will develop a realtime vision system based on FPGA's and will evaluate the performance of the developed FPGA-based vision system experimentally. First, we will describe the developed FPGA-based vision system. Secondly, we will implement three vision algorithms on the vision system; computation of the image gravity center, detection of object orientation using a radial projection, and the computation of Hough transform. We will then execute these algorithms on the FPGA-based realtime vision system.

I. Introduction

In this article, we will develop a realtime vision system based on FPGA's. The goal of this article is to demonstrate the performance of the developed FPGA-based vision system experimentally.

Current vision systems are categorized into two; software-based approach and ASIC-based approach. Vision algorithms are implemented on programs, which are performed on a general-purpose MPU, in a software-based approach. This approach can perform various algorithms and can construct a system in a low cost. On the other hand, it often requires much computation time, resulting in failure of the realtime processing. Logic circuits specialized to individual vision algorithms are designed and are implemented on LSI's in an ASIC-based approach. Many vision algorithms have been implemented on ASIC's including Fourier transforms[1], Hough transforms[2], normalized correlations[3, 4], and stereo vision algorithms[5]. In addition, VLSI's composed of logic circuits and analog sensor circuits have been proposed[6, 7]. Fast computation can be performed in these ASIC-based approach but it requires huge time and cost to design and to implement logic circuits on ASIC's. Consequently, software-based approach has good flexibility but lacks realtimeness while ASIC-based approach has good realtimeness but lacks flexibility and cost efficiency. To overcome this dilemma, we will develop a vision system based on FPGA's. FPGA's are special VLSI's where users can configure logic circuits. We can design a logic circuit specialized to a vision algorithm and can implement the designed logic circuit on an FPGA. This implies that realtimeness of a vision algorithm can be realized by implementing the algorithm in a logic circuit and flexibility of a vision system can be realized by redesigning and reconfiguring the logic circuit on an FPGA.

In this article, we will develop an FPGA-based realtime vision system. First, we will describe the developed FPGA-based vision system. Secondly, we will implement three vision algorithms on the vision system; computation of the image gravity center, detection of object orientation using a radial projection, and the computation of Hough transform. We will then perform these algorithms on the FPGA-based realtime vision system.

II. FPGA-based Realtime Vision System

A. Hardware

In this section, we will describe a vision system based on FPGA's. In many applications including object handling and human identification, vision systems are required to process successive images in realtime. In addition, vision systems must be flexible to cope with various environments. We will develop an FPGA-based vision system to achieve both realtimeness and flexibility. FPGA's are special VLSI's in which users can configure logic circuits. We can design a logic circuit specialized to a vision algorithm and implement the designed logic circuit on an FPGA. This implies that the realtimeness of a vision system can be realized by implementing a vision algorithm in a logic circuit. Also, flexibility of a vision system can be realized by redesigning and reconfiguring the logic circuit on an FPGA. Note that many vision algorithms can be performed in parallel. Thus, implementing these algorithms on FPGA's can reduce the computation time.

Figure 1 shows a prototype of the FPGA-based vision system. This prototype consists of an FPGA, a video decoder, a video encoder, SRAM, and a PC interface. An FPGA, Xilinx Vertex 2000-E, is mounted on an FPGA board, SK Electronics CM69. Two million gates can be implemented on the FPGA. Video decoder

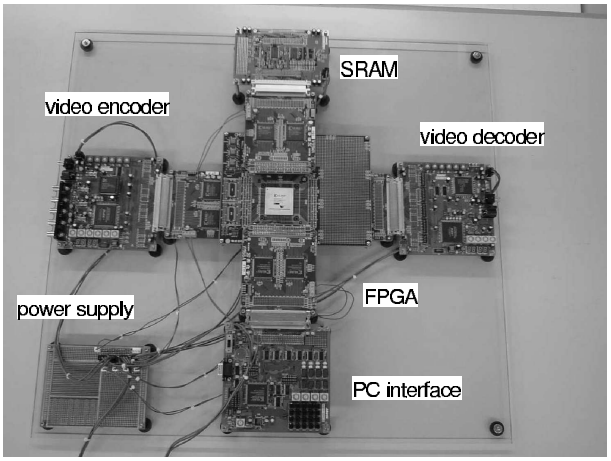


Fig. 1. Prototype of FPGA-based vision system

board Mitsubishi Electric Micro-Computer Application Software MU200-VDEC and video encoder board MU200-VENC involves video decoding/encoding chips, Brooktree Bt812 and Bt856. SRAM board, MU200-SRAM, offers 1Mbyte memory. Successive images from a CCD camera in NTSC format are captured by a video decoder and are sent to the following image processing circuits. Image processing circuits are implemented on the FPGA. Data during the image processing can be stored in SRAM if necessary. Processed images are encoded to NTSC format in a video encoder to display the images on a video monitor. Image processing circuits are designed and are synthesized on a PC. Designed logic circuits are downloaded to the FPGA through a PC interface, MU200-EX40.

B. C/C++-based Design of Logic Circuits

We will introduce C/C++-based description for the design of logic circuits of vision algorithms. In the past decades, many vision algorithms have been developed using C/C++ language. Thus, description of logic circuits in C/C++ language connects the development of algorithms to the design of logic circuits seamlessly. In addition, we have to repeat the description of a logic circuit and its verification during the design of a vision processing circuit. Designing of logic circuits in C/C++ language can hasten this repetition, which reduces the time for the design. Thus, we have introduced the design of logic circuits in C/C++ language.

We have introduced SystemCompiler, which can translate C/C++ description into HDL description. The SystemCompiler offers the circuit design in CycleC. CycleC is a subset of C/C++. Any logic design in CycleC can be compiled by a C/C++ compiler and can be executed on a computer. Due to this property, we can verify a designed logic circuit in a short time. Logic design in CycleC can be transformed into the design in hardware description languages including VHDL and VerilogHDL.

TABLE I
DESCRIPTION OF SELECTOR IN SYSTEMCOMPILER

```
class Selecter {
public:
  uint1 reg;
  Selecter ( void ){ reg = 0;}
  void run ( uint1, uint1,
       uint1, uint1, uint1, uint1& );
};

void Selecter::run(uint1 clk, uint1 rst,
     uint1 a, uint1 b, uint1 s, uint1&y)
{
  uint1 temp = s ? a : b;
  if ( infer_clock(clk) ||
       infer_reset(!rst) ) {
    if (!rst) { reg = 0; }
    else { reg = temp; }
    y = reg;
  }
}
```

Design in hardware description languages can be synthesized and be implemented on an FPPA. Consequently, we can develop vision algorithms and can implement them on FPGA's in an efficient manner using SystemCompiler. Table I shows a description of a selector in CycleC. As described in the table, logic circuits can be described using classes and class methods in C/C++ language.

III. IMPLEMENTING VISION ALGORITHMS

We have implemented three vision algorithms on the FPGA-based vision system; 1) computation of the image gravity center, 2) detection of object orientation using a radial projection[8], and 3) the computation of Hough transform.

A. Computation of Image Gravity Center

First, we will describe the implementation of the computation of the image gravity center. Let us briefly review the gravity center of an image. Let W and H be the width and the height of a grayscale image and $g(x, y)$ be the pixel value at lattice point (x, y). Then, the gravity center of an image is given by

$$\begin{bmatrix} c_x \\ c_y \end{bmatrix} = \begin{bmatrix} W_x/W \\ W_y/W \end{bmatrix} \tag{1}$$

where

$$\begin{bmatrix} W_x \\ W_y \end{bmatrix} = \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} \begin{bmatrix} x \\ y \end{bmatrix} g(x, y),$$

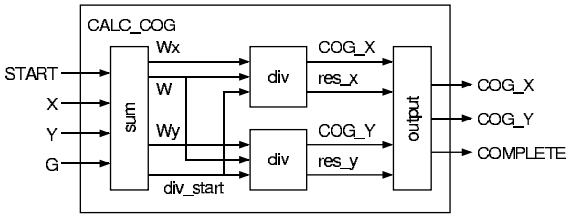
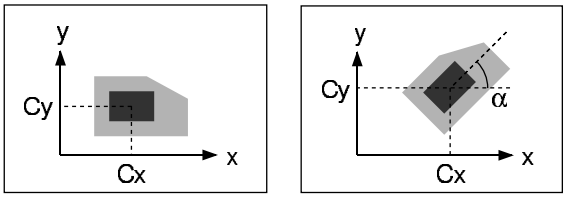


Fig. 2. Circuit to compute gravity center



(a) template image (b) input image

Fig. 3. Image with translation and rotation

$$W = \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} g(x,y).$$

The computation of the image gravity center requires three summations and two divisions. Note that the computation of summations W_x, W_y, and W can be performed along the capturing process of an image. Namely, the summations are completed when the capturing process finishes. Then, two divisions are performed after the capturing process. Fortunately, the divisions can be performed in parallel. Consequently, the computation of the gravity center finishes in the time needed to one division after the capturing process. Figure 2 illustrates the logic circuit to compute the gravity center. This involves four input signals, `START`, `X`, `Y`, and `G`, and three output signals, `COG_X`, `COG_Y`, and `COMPLETE`. Signal `START` triggers the computing, signals `X` and `Y` denote the coordinates of a pixel, and signal `G` represents the pixel value at point (x, y). Signals `COG_X` and `COG_Y` denote the coordinates of the gravity center and signal `COMPLETE` informs the completion of the computing. Module `SUM` computes W_x, W_y, and W given in the above equation along the capturing process of an image. After the summations complete, signal `div_start` triggers the divisions.

B. Radial Projection

Second, we will describe the implementation of the computation of radial projection. Let us briefly review the radial projection. Let $O-xy$ be a coordinate frame fixed on a grayscale image and $g(x, y)$ be the pixel value at point (x, y) on the image. Let us integrate the pixel value along a half line from the image gravity center. Let θ be the angle of a half line from the x-axis. Then, the integral depends one parameter θ and can be described as follows:

$$R(\theta) = \int_0^\infty g(c_x + \xi\cos\theta, c_y + \xi\sin\theta)d\xi, \qquad (2)$$

where c_x and c_y denote the coordinates of the image gravity center. A set of integral $R(\theta)$ over $[0, 2\pi]$ is referred to as *radial projection*.

Let $g_{template}$ and g_{input} be the template and the input images. Let α be the rotational angle of an object in the input image, as illustrated in Figure 3. Let $R_{template}$ and R_{input} be their radial projections. Projections $R_{template}$ and R_{input} then satisfy the following equation:

$$R_{template}(\theta) = R_{input}(\theta + \alpha), \quad \forall\theta. \qquad (3)$$

Let us introduce an error function between the projections:

$$S(\tau) = \int_0^{2\pi} |R_{template}(\theta) - R_{input}(\theta + \tau)|\, d\theta. \qquad (4)$$

This error function reaches to its minimum value 0 at $\tau = \alpha$. Consequently, we can find rotational angle α by minimizing the above error function.

The computation of the projection given in eq.(2) cannot be started before the capturing process has completed. Namely, the computation cannot be performed during the capturing process. This may hinder the realtime image processing. Thus, we will introduce the concept of *voting* in Hough transform so that the computation can be performed along the capturing process. From eq.(2), we have

$$x = c_x + \xi\cos\theta,$$
$$y = c_y + \xi\sin\theta.$$

TABLE II
VOTING IN COMPUTATION OF PROJECTION

initialize array $R(\theta)$
for $(x, y) = (0,0), (0,1), \cdots, (W-1, H-1)$ do
  $\Delta x = x - c_x$, $\Delta y = y - c_y$
  compute $\theta = \arctan(\Delta y/\Delta x)$
  increase $R(\theta)$ by pixel value $g(x, y)$
end

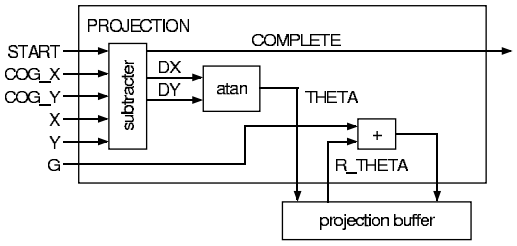


Fig. 4. Circuit to compute projection

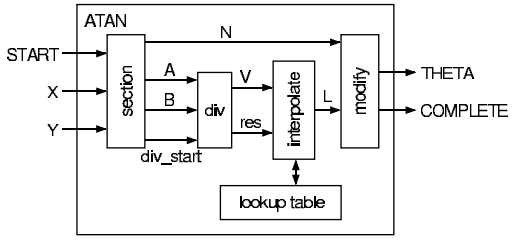


Fig. 5. Circuit to compute function *atan*

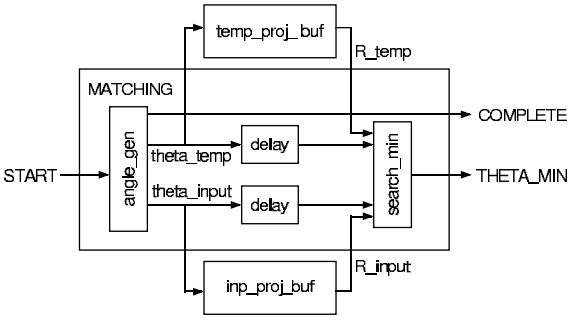


Fig. 6. Circuit to compare projections

Eliminating parameter ξ yields

$$\tan\theta = \frac{y - c_x}{x - c_y}. \tag{5}$$

This describes the contribution of the pixel value at point (x, y) to the integral along a half line specified by θ. Namely, pixel value $g(x, y)$ should be added to the integral $R(\theta)$, where angle θ is given by the above equation. Recall that pixel values are obtained in sequence in the capturing process of an image. Thus, every pixel value can be added to an appropriate integral during the capturing process. This voting process is summarized in Table II. Figure 4 illustrates the logic circuit to compute a projection using the voting. This involves six input signals, `START`, `COG_X`, `COG_Y`, `X`, `Y`, and `G`, and one output signal `COMPLETE`. Module `subtracter` computes $\Delta x = x - c_x$ and $\Delta y = y - c_y$. Module `atan` calculates $\theta = \arctan(\Delta y/\Delta x)$. Projection $R(\theta)$ is stored in a projection buffer. Integral value at angle θ is loaded from the projection buffer by specifying angle θ, which corresponds to signal `THETA`. The integral is increased by pixel value $g(x, y)$, which is given by signal `G`, and is stored back to the projection buffer.

Module `atan` is described in Figure 5. Let us select representative values v in interval $[0, 1]$ and compute $\theta = \arctan(v)$. Pairs of v and θ at representative values are stored in a lookup table in advance. Let us design a circuit to compute $\arctan(y/x)$ at point (x, y) that satisfies $0 \le y \le x$. First, compute y/x, which in involved in interval $[0, 1]$. Then, function value $\arctan(y/x)$ can be calculated through the linear interpolation between representative values in the lookup table. Module

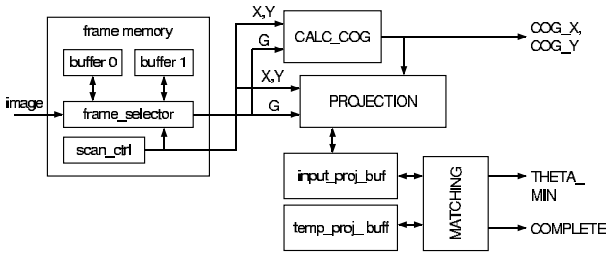


Fig. 7. Circuit to detect planar motion

`interpolate` performs this interpolation process. Function value at any point (x, y) can be computed using the above interpolation process. For example, assume that point (x, y) satisfies $0 \le -x \le y$. Note that function value at point (x, y) is given by $\pi - \arctan(-x/y)$. Since quotient $(-x/y)$ is involved in interval $[0, 1]$, module `interpolate` can compute the value of $\arctan(-x/y)$. Let us divide the x-y plane into 8 sections by signs of x and y as well as whether $|x|$ is smaller than $|y|$ or not. Then, we can compute the value of $\arctan(y/x)$ at any point using module `interpolate`. Module `section` determines the section. Signal `N` specifies the section number, signal `A` denotes the smaller value of $|x|$ and $|y|$, and signal `B` denotes the other. Module `modify` builds the function value $\arctan(y/x)$ from $\arctan(A/B)$ and N.

Figure 6 illustrates the logic circuit that performs the minimization of the error function given in eq.(4). Module `angle_gen` generates two signals `theta_temp` and `theta_input`. The former denotes angle θ and the latter represents angle $\theta + \tau$ in eq.(4). Corresponding projection values are loaded form buffers `temp_proj_buf` and `inp_proj_buf`. Module `search_min` sums up the absolute difference between the projection values and finds angle α that minimizes the calculated sum.

When the object image can be separated from its background, we can detect the position of a planar motion object by computing the image gravity center. In addition, we can detect its orientation by comparing radial projections of the template and the input images. Figure 7 illustrates the circuit to detect the position and the orientation of a planar motion object based on the image gravity center and the comparison between radial projections. Double buffering is applied to the video decoder board to capture images successively. Module `frame_selector` alternates the two buffers. Module `scan_ctrl` generates the successive coordinates (x, y) that cover the whole image. Projection of the template image is computed and is stored in the template projection buffer in advance. Signals `COG_X` and `COG_Y` represent the position of a planar motion object and signal `THETA_MIN` denotes its rotation.

TABLE III
VOTING IN COMPUTATION OF HOUGH TRANSFORM

initialize array $H(\rho, \theta)$
for $(x, y) = (0, 0), (0, 1), \cdots, (W-1, H-1)$ do
  for $k = 0, 1, \cdots, K-1$ do
    compute $\theta = k\Delta\theta$
    compute $\rho = (-\sin\theta)x + (\cos\theta)y$
    increase $H(\rho, \theta)$ by pixel value $g(x, y)$
end

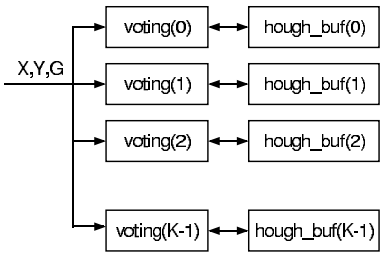


Fig. 8. Circuit to compute Hough transform in parallel

C. Hough transform

Hough transform is one of fundamental vision algorithms and is applied to various applications. For example, the position and the orientation of a planar motion object can be detected robustly against illumination change and occlusion using Hough transform[9]. Hough transform requires much computation time but involves high parallelism. This implies that implementing Hough transform on an FPGA yields fast computation. In this section, we will implement Hough transform on an FPGA-based realtime vision system.

Hough transform is a mapping from a discrete image $g(x, y)$ to Hough space $H(\rho, \theta)$. Let us first compute pairs of ρ and θ that satisfy the following equation at each lattice point (x, y):

$$(-\sin\theta)x + (\cos\theta)y = \rho.$$

Then, increase the value of $H(\rho, \theta)$ by pixel value $g(x, y)$ at any pair of ρ and θ that satisfies the above equation. This operation is referred to as *voting*. Performing the voting at all lattice point in a discrete image yields the Hough transform of the discrete image. The above equation represents a group of lines that intersect point (x, y). Parameter ρ describes the signed distance between a line and the origin and θ denotes the angle between the line and x-axis. Computation process of Hough transform is summarized in Table III.

Fig. 8 sketches a circuit to compute Hough transform. Control signals are omitted from the figure. Let us divide interval $[0, 2\pi]$ into K sections and let $\Delta\theta = 2\pi/K$. Then,

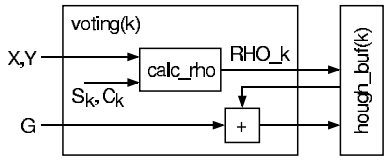


Fig. 9. Circuit to perform voting

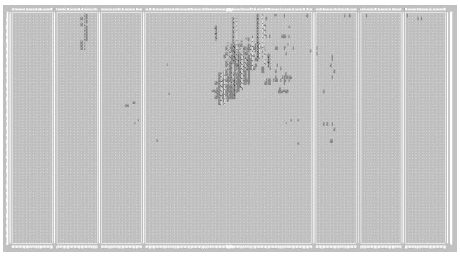


Fig. 10. Implementation of logic circuit to compute gravity center

angle θ takes discrete values; $k\Delta\theta$ $(k = 0, 1, \cdots, K-1)$. Computation corresponding to angle $k\Delta\theta$ is performed by a module `voting(`k`)`, which is illustrated in Fig. 9. Result of Hough transform at angle $k\Delta\theta$ is stored in a buffer `hough_buf(`k`)`. Input signals `X` and `Y` denote the coordinates of a lattice point and `G` represents its pixel value. Module `voting(`k`)` contains value of $c_k = \cos(k\Delta\theta)$ and $s_k = \sin(k\Delta\theta)$. Module `calc_rho` compute the value of $\rho_k = c_k y - s_k x$, which corresponds to signal `RHO_k`. After finishing the computation of distance ρ_k, buffer `hough_buf(`k`)` is updated accordingly. First, specifying signal `RHO_k`, integral value corresponding to distance ρ_k is loaded from the buffer. The integral value is increased by pixel value $g(x, y)$, which is given by signal `G`. Then, the updated integral value is restored in the buffer. The above computation can be performed in parallel for angles $k\Delta\theta$ $(k = 0, 1, \cdots, K-1)$. Consequently, Hough transform can be performed by a circuit consisting of modules `voting(0)` through `voting(`$K-1$`)` and buffers `hough_buf(0)` through `hough_buf(`$K-1$`)` connected in parallel as illustrated in Fig. 8.

IV. EXPERIMENTAL EVALUATION

A. Computation of Image Gravity Center

We have implemented the logic circuit to compute the image gravity center on the FPGA-based realtime vision system. The circuit computes the image gravity center of 256×256[pixel] region at the center of an input image. FPGA consumes 2% of total gates. Figure 10 shows the gates in the FPGA consumed to implement the logic circuit. It turns out that the circuit can be driven at 58MHz in maximum. Here, we will drive the circuit at 12MHz, which is equal to the frequency of image capturing. Since

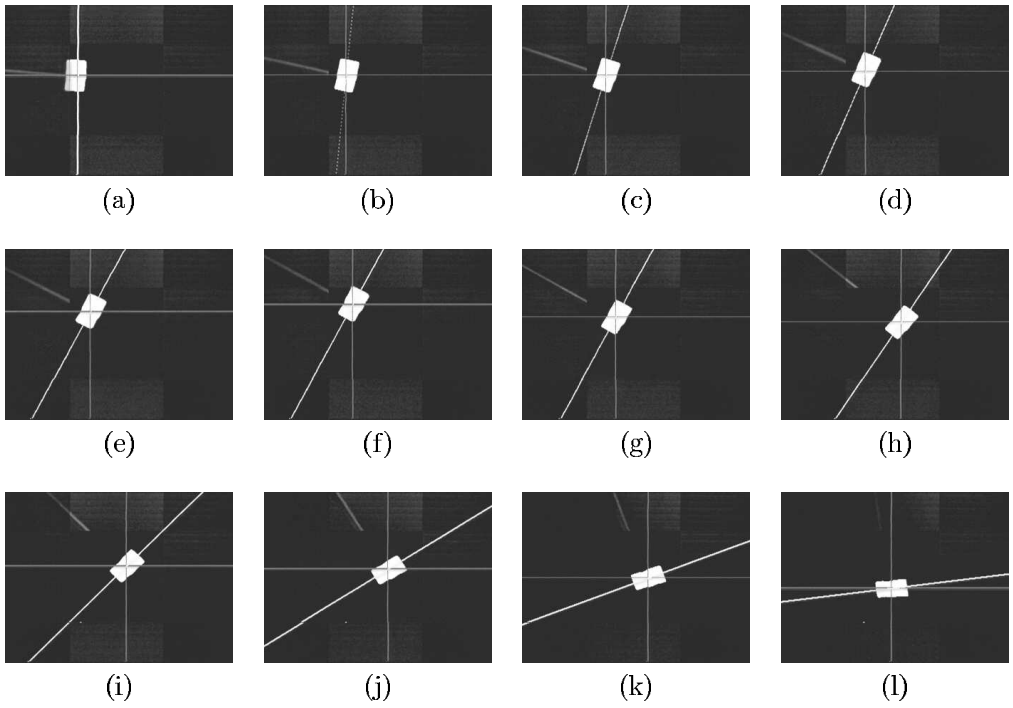


Fig. 13. Detection of position and orientation

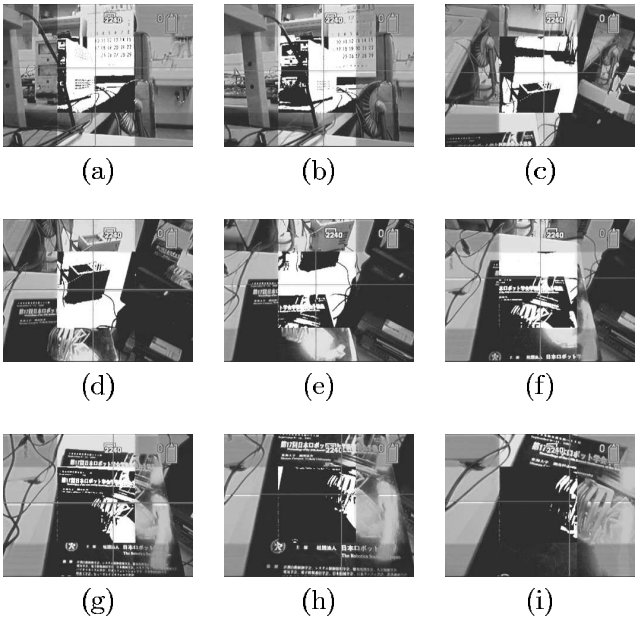


Fig. 11. Detection of image gravity center

one division requires 32 clocks, the computation of the image gravity center finishes in about 2.7[μsec] after the image capturing. Figure 11 describes a result of the computation of the center. Vertical and horizontal lines in each image indicate the position of the center. It turns out that the gravity center of successive images can be computed appropriately in realtime.

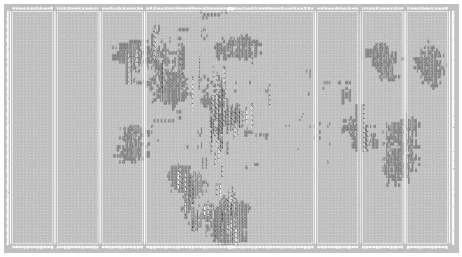
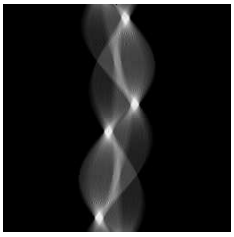


Fig. 12. Implementation of logic circuit to detect planar motion object

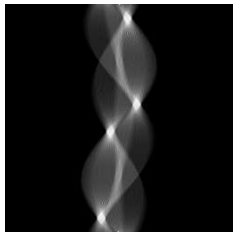
B. Radial Projection

We have implemented the logic circuit to detect the planar motion of an object on the FPGA-based realtime vision system. The circuit computes the image gravity center and the radial projection of 256×256[pixel] region at the center of an input image. Rotational angle is expressed by 64 steps. FPGA then consumes 13% of total gates. Figure 12 shows the gates in the FPGA consumed to implement the logic circuit. It turns out that the circuit can be driven at 24MHz in maximum. Here, we will drive the circuit at 12MHz, which is equal to the frequency of image capturing. Since the voting for one pixel requires 1 clocks, the computation of a radial projection needs 1×256×256 clocks, which corresponds to 5.46[msec]. The computation of a radial projection finishes simultaneously with the capturing process of an image since the computation is performed in parallel along the image capturing.

(a) input



(b) program



(c) simulation

Fig. 14. Hough transforms by software and by simulation of logic circuit

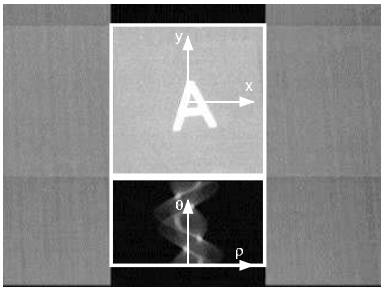


Fig. 15. Image and its Hough transform

Since matching between two radial projections requires 64×64 clocks, the detection of the position and the orientation finishes in about 341[μsec] after the image capturing. Figure 13 demonstrates the computation of the position and the orientation of a planar motion object. Vertical and horizontal lines in each image indicate the computed position of an object and the slanted line describes its detected rotation. The successive images have shown that the position and the orientation of an object are successfully detected using the FPGA-based realtime vision system.

C. Hough transform

We have implemented the logic circuit to compute the Hough transform of an input image on the FPGA-based realtime vision system. The transform is stored in block RAM's; specialized regions for memory in an FPGA. We will describe the circuit by CycleC and will verify the description through simulation. Fig. 14 summarizes the verification by simulation. Fig. 14-(a) shows the Hough transform computed by a C program while Fig. 14-(b) describes the transform obtained by the simulation of the

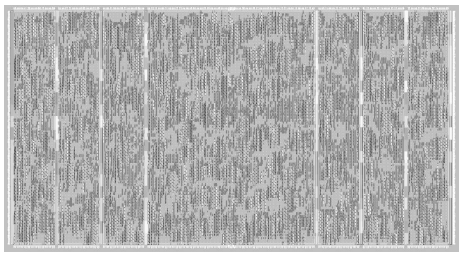


Fig. 16. Implementation of logic circuit to compute Hough transform

circuit. Both transforms coincide with each other well and we conclude that the description by CycleC can compute the Hough transform. Fig. 15 shows the output of the Hough transform circuit. The circuit computes the Hough transform of 256×256[pixel] region at the center of an image and displays the result below. Angle θ takes 144 discrete values at the intervals of 2.5°. Signed distance ρ takes a discrete value between −127[pixel] and 127[pixel] at the intervals of 1[pixel]. Then, the FPGA consumes 77% of total gates. Figure 16 shows the gates in the FPGA consumed to implement the logic circuit. The circuit can be driven at 24MHz. Since the voting for one pixel requires 2 clocks, the computation of Hough transform needs 2×256×256 clocks, which corresponds to 5.46[msec]. The computation finishes simultaneously with the capturing process of an image since the computation is performed in parallel along the image capturing. Fig. 17 shows the computation of Hough transform. As shown in the figure, the FPGA-based vision system can compute Hough transforms of successive images in realtime.

V. Concluding Remarks

In this article, we have described an FPGA-based realtime vision system and three vision algorithms implemented on the system. We have implemented an algorithm to detect the position and the orientation of a planar motion object using image gravity center and radial projection. It turns out that the algorithm consumes 15% gates out of 2 million gates in an FPGA. The algorithm can be performed in about 5[msec]. Note that the algorithm can be performed in parallel along the capturing of an image and finishes in about 0.3[msec] after the image capturing. We have also implemented the computation of Hough transform. We find that it consumes 80% of gates out of 2 million dates in an FPGA. This consumption is caused by the implementation of buffers in an FPGA. Implementing buffers in SRAM's instead of in an FPGA reduces the consumption of gates. The computation can be performed in about 5[msec]. Note that The computation finishes simultaneously with the capturing process of an image. Consequently, we can reduce the latency as well as the computation time by implementing vision

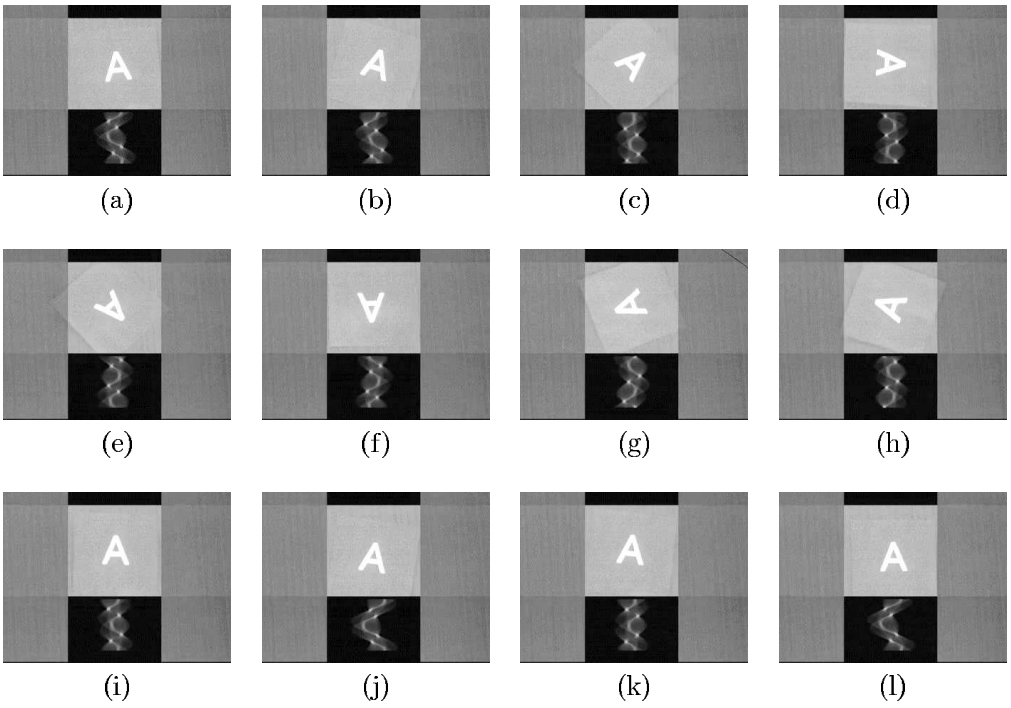


Fig. 17. Realtime computation of Hough transform

algorithms on the FPGA-based realtime vision system.

We have implemented all of a circuit including buffers on an FPGA. By implementing buffers in external SRAM's, we can implement more complex algorithms. Matched filter[10], phase-only correlation method[11], Radon-Fourier transformation method, and generalized Hough transform[12] can detect the position and the orientation of a planar motion object robustly against the change of illumination and occlusion. Unfortunately, these algorithms need operations such as 2D/1D Fourier transform, Radon transform, and polar transform, which require much computation time. This hinders the realtime execution of these algorithms. Implementing these algorithms on the FPGA-based vision system can break this barrier. We will implement these algorithms on the FPGA-based vision so that they can be performed in realtime.

Acknowledgements

This research was supported from the New Energy and Industrial Technology Development Organization in 2001.

References

[1] Thompson, C. D., *Fourier Transforms in VLSI*, IEEE Trans. on Computers, Vol.C-32, No.11, pp.1047–1057, 1983

[2] Maresca, M., Lavin, M., and Li, H., *Parallel Hough Transform Algorithms on Polymorphic Torus Architecture*, Levialdi, S. eds., *Multicomputer Vision*, Academic Press, pp.9–21, 1988

[3] Inoue, H., Tachikawa T., and Inaba, M., *Robot Vision System with a Correlation Chip for Real-time Tracking, Optical Flow and Depth Map Generation*, Proc. IEEE Int. Conf. on Robotics and Automation, pp.1621–1626, Nice, May, 1992

[4] Bugeja, A. and Yang, W., *A Reconfigurable VLSI Coprocessing System for the Block Matching Algorithm*, IEEE Trans on VLSI Systems, Vol.5, No.3, pp.329–337, 1995

[5] Hariyama, M., Takeuchi, T., and Kameyama, M., *VLSI Processor for Reliable Stereo Matching Based on Adaptive Window-Size Selection*, Proc. 2001 IEEE Int. Conf. on Robotics and Automation, pp.1168–1173, Seoul, May, 2001

[6] Eklund, J.-E., Svensson, C., and Aström, A., *VSLI Implementation of a Focal Plane Image Processor – A Realization of the Near-Sensor Image Processing Concept*, IEEE Trans. on VLSI Systems, No.4, Vol.3, pp,322–335, 1996

[7] Ishii, I., Nakabo, Y., and Ishikawa, M., *Target Tracking Algorithm for 1ms Visual Feedback System using Massively Parallel Processing Vision*, Proc. 1996 IEEE Int. Conf. on Robotics and Automation, pp.2309–2314, Minneapolis, May, 1996

[8] Tsuboi, T., Masubuchi, A., Hirai, S., Yamamoto, S., Ohnishi, K., and Arimoto, S., *Video-frame Rate Detection of Position and Orientation of Planar Motion Objects using One-sided Radon Transform*, Proc. IEEE Int. Conf. on Robotics and Automation, Vol. 2, pp.1233–1238, Seoul, May, 2001

[9] Onishi, H. and Suzuki, H., *Detection of Rotation and Parallel Translation Using Hough and Fourier Transforms*, Proc. 1996 Int. Conf. on Image Processing, Vol.3, pp.827–830, 1996

[10] Turin, G. L., *An Introduction to Digital Matched Filters*, Proceedings of the IEEE, Vol. 64, No. 7, pp.1093–1112, 1977

[11] Chen, Q., Defries, M, and Deconinck, F., *Symmetric Phase-Only Matched Filtering of Fourier-Mellin Transforms for Image Registration and Recognition*, IEEE Trans. PAMI, Vol.16, No.12, pp.1156–1168, 1994

[12] Ballard, D., H., *Generalizing the Hough Transform to Detect Arbitrary Shapes*, Pattern Recognition, Vol. 13, No. 2, pp.111–122, 1981