An Autonomous Flying Object Navigated by Real-time Optical Flow and Visual Target Detection

Hitoshi Yamada, Takashi Tominaga, and Michinori Ichikawa
Laboratory for Brain-Operative Devices, RIKEN BSI,
2-1 Hirosawa, Wako, Saitama, Japan 351-0198
ymd@brain.riken.go.jp

Abstract

We have implemented logic, including real-time optical flow and target detection, using four CMOS video cameras and FPGA mounted on an indoor autonomous flying object. The two techniques presented here, recursive integration and masking optical flow, enhance the signal-to-noise ratio and accuracy of orientation of optical flow. The optical flow of the images from three cameras set to provide a horizontal filed of view can be used to derive the flying object's attitude in terms of roll, pitch, yaw, and height. A target detection process using a camera mounted on top of the flying object can also be used to derive the attitude. These visual processing and nonlinear controlling processes were all implemented in an FPGA (Xilinx XC2V1500). The flying object described here could hover autonomously for several minutes.

1. Introduction

Exploring biologically inspired vision processing is the first step in obtaining an understanding of the brain that will eventually allow the creation of a brain-type computer. A previous biological study [1] indicated that the housefly, *Musca domestica*, effectively uses the optical flow detected from their compound eyes. The first step toward our goal of developing a system to allow navigation of an indoor autonomous flying object using only visual input is to mimic the fly eye's function. Key points that must be resolved to realize this are compactness and sufficient power of calculations. For calculation of the attitude and location of the flying object, all processes should be performed with a CMOS sensor frame rate of 25 *msec*. Methods of calculating optical flow should be tested in real-time experiments. The total weight of the controller should be within 100 g for an indoor flying object of 50 cm. A field programmable technology, such as FPGA, would be suitable for the development of this kind of controller, because of its compactness, calculation speed, and field programmability. We have implemented a high-speed visual processing controller of optical flow and target detection with wide fields of view using four CMOS sensors mounted on the flying object. In previous studies, we have implemented some of the functions of the compound eye using modern digital processing technology realized in FPGA [2], [3].

Images from three CMOS sensors set horizontally provide the flying object's attitude as the derivatives d(Roll)/dt, d(Pitch)/dt, d(Yaw)/dt, and d(Height)/dt through real-time optical flow calculations. In these calculations we introduce two techniques for enhancing the accuracy of orientation of the optical flow and its preprocessing. Another CMOS sensor mounted on top of the flying object detects the locations of two targets on the ceiling to calculate the absolute location of the flying object, and is also used to derive the attitude in terms of roll, pitch, yaw, and height. We will introduce an application of these real-time calculations and flight demonstration using FPGA technology.

2. System Overview

We constructed a flying object with a custom cross-shaped carbon chassis (length 53 cm) mounted with four CMOS cameras, a control unit, and four DC motors. First, the configuration of the FPGA should be downloaded from a PC, and then several parameters of PID-based control are set *via* a parameter setting board. The total weight of the flying object is about 400 g, while maximum lifting power is about 500 g through the four motors. The object flies under the power of a DC 7.2V 10A wiring power supply during flight.

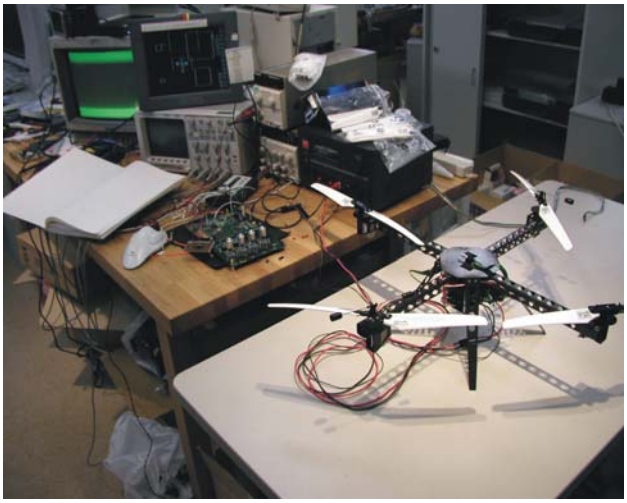


Fig.1 Photo of the system from above

2.1 Electronics

The control unit consists of three parts: the main FPGA, a debugging board, and power units. The purpose and details of each are described below.

Main FPGA Unit

Aim: High-speed image processing, determining control parameters, communication with peripherals
FPGA: Xilinx XC2V1500FG676;
PCB: 53 mm×74 mm;
Weight: 30 g (including components);
Number of Used Slices: 6241 (/7680 81%);
Number of Used Block RAM's: 28 (/48 50%);
Power Consumption:
4.32W (7.2V 0.6A) after configuration;

Debugging Unit

Aim: transform debugging images to NTSC signals
FPGA: XC2S100E;
PCB: 53 mm×74 mm;
Weight: 21 g (including components);
Number of Used Slices: 95 (/3072 3%);

Power Unit

Aim: Supply stabilized DC 3.3V to Main and Debugging Boards, Power control of motor
FET: Toshiba 2SK2493 ×8;
PCB: 53 mm×74 mm;
Weight: 20 g;

CMOS sensor

Aim: Image capturing
Image Sensor: OmniVision OV6630;
PCB: Olympus (Komasa);
Weight: 1.0 g;
Array Element: (CIF) 352 (H) × 288 (V);
Frame speed: 40 FPS;
Scan: Progressive Scan;
Field of View: 60° (H);
Resolution: 0.17°/pixel (horizontally);
Power supply: 3.3V;
Dimension (Komasa): 10 mm (W) × 20 mm (D) × 6.6 mm (H);

Motor: Mabuchi FK180SH;
72W (7.2V10A) while hovering;

Parameter Setting Board:

Aim: Send 28 controlling constants to the main FPGA unit
FPGA: XCV300PQ240
PCB: 130 mm × 230 mm
Number of Used Slices: 1500 (/3072 48%)
Power Supply: 2.4W (6V 0.4A)

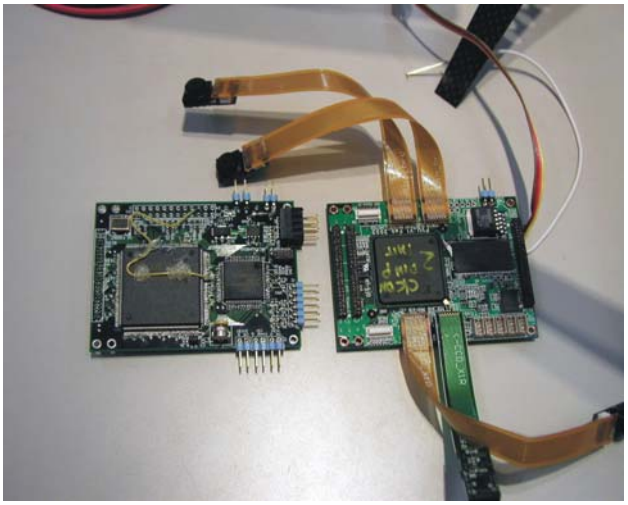


Fig2. Photo of the debugging board, the main FPGA unit, and four CMOS sensors

3. Optical Flow Detection

Three CMOS sensors, *i.e.*, those set with a horizontal field of view, independently derive each optical flow as moving vectors $\vec{M}_i = (M_{ix}, M_{iy})$, where *i=0, 1,* and *2*, corresponding to the front, left, and right sensor, respectively. Each moving vector is complementarily integrated and derives the attitude of the flying object as derivatives of ***Roll***, ***Pitch***, ***Yaw***, and ***Height***. Three horizontal sensors that do not look backward were used to avoid disturbance by people moving behind the flying object. The CMOS sensor (Olympus Komasa / OmniVision OV6630) has a 60° field of view at 40 frames per second.

3.1 Image Preprocessing

We derived optical flow from images from a CMOS sensor stored in frame memory with a subtraction process between pairs of images: $Y_f(x,y,t)$ and $Y_s(x,y,t)$. The former, $Y_f(x,y,t)$, represents the present image, while the latter represents the previous image. These are accumulated images with a proper time constant for each, and are derived from the brightness of the CMOS sensor images depending on the following equations:

$$Y_f(x,y,t) = \sum_{n=k}^{t} \{\frac{3}{4} Y_{raw}(x,y,n) + \frac{1}{4} Y_f(x,y,n-1)\}$$

$$Y_s(x,y,t) = \sum_{n=k}^{t} \{\frac{1}{4} Y_{raw}(x,y,n) + \frac{3}{4} Y_s(x,y,n-1)\}$$

where

$Y_{raw}(x,y,n)$: raw images

n= 1, 2, 3…, t: frame number

x=1, 2, 3…, 352: horizontal pixel number

y=1, 2, 3…, 288: vertical pixel number

$$Y_f(x,y,0) = Y_s(x,y,0) = 0\,.$$

These two images can be easily implemented in hardware accumulating raw images to be added recursively to stored images. Each fraction of the adding operation leads to each relaxation coefficient and determines the percentage of the raw image within the accumulated image. The fraction governs the distribution of each frame, which represents each time domain. Thus, we call this the *recursive integration* method. This method allows smoothing and enhancement of the signal-to-noise ratio, because the operation is analogous to averaging or integration. As we introduced this as a preprocess in deriving optical flow, our system has great robustness to high frequency disturbance of the movement of the flying object.

3.2 Masking Optical Flow

The conventional optical flow method includes essential aliases, which are caused by the skew lines. That is, moving of the skew line influences both horizontal and vertical movement of the moving vector. We overcame this problem ignoring these skew lines using the following masking technique (patent pending). Optical flow is derived through a subtraction operation between the two images mentioned above. During the subtraction process, we used modified images shifted in four directions horizontally and vertically. We could determine the direction of movement in subtraction of these shifted images from non-shifted images and counting the pixels of the subtracted images in each direction. However, the subtracted images still include skew line aliases. We felt that the skew line should be neglected using proper masks. Such masks can be implemented easily with combinations of shifted images. The precise expressions are as follows.

First the two images $Y_f(x,y,t)$ and $Y_s(x,y,t)$ are used to determine the un-normalized moving vector $\vec{m} = (m_1, m_2)$, which still includes the skew aliases. We shifted these images, subtracted them from each other, and counted the number of pixels exceeding some threshold level.

$$D_{fs}^{HP}(t) = |\,Y_f(x,y-1,t) - Y_s(x,y,t)\,|$$
$$D_{fs}^{DP}(t) = |\,Y_f(x-1,y,t) - Y_s(x,y,t)\,|$$
$$D_{fs}^{EP}(t) = |\,Y_f(x,y,t) - Y_s(x,y,t)\,|$$
$$D_{fs}^{FP}(t) = |\,Y_f(x+1,y,t) - Y_s(x,y,t)\,|$$
$$D_{fs}^{BP}(t) = |\,Y_f(x,y+1,t) - Y_s(x,y,t)\,|$$

This shifting matrix:

$$\begin{bmatrix} - & D_{fs}^{BP}(t) & - \\ D_{fs}^{DP}(t) & D_{fs}^{EP}(t) & D_{fs}^{FP}(t) \\ - & D_{fs}^{HP}(t) & - \end{bmatrix}$$

can be easily realized in hardware using line buffer memory in addition to the frame memory.

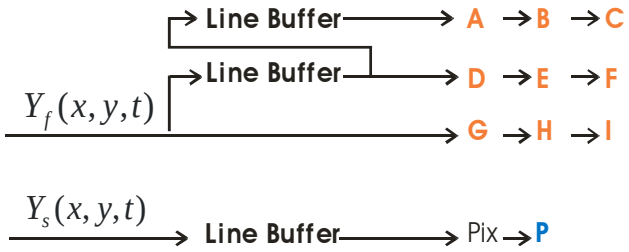


Fig.3 Schematic of line buffer for shifting matrix

Thus, the un-normalized moving vector could be derived as follows:

$$m_1 = \frac{1}{2}\{D_{fs}^{DP}(t) + D_{fs}^{FP}(t)\}$$
$$m_2 = \frac{1}{2}\{D_{fs}^{HP}(t) + D_{fs}^{BP}(t)\}$$

This un-normalized moving vector still includes the skew line aliasing noise. Then, we introduce the following masks, which neglect the skew line for compensation of movement of the horizontal or vertical directions. The masks are easily implemented in hardware realizing the following equation:

$$D'^{HP}_{fs}(t) = I[D_{fs}^{DP}(t)]I[D_{fs}^{FP}(t)]B[D_{fs}^{HP}(t)]D_{fs}^{EP}(t)$$
$$D'^{DP}_{fs}(t) = I[D_{fs}^{HP}(t)]I[D_{fs}^{BP}(t)]B[D_{fs}^{DP}(t)]D_{fs}^{EP}(t)$$
$$D'^{FP}_{fs}(t) = I[D_{fs}^{HP}(t)]I[D_{fs}^{BP}(t)]B[D_{fs}^{FP}(t)]D_{fs}^{EP}(t)$$
$$D'^{BP}_{fs}(t) = I[D_{fs}^{DP}(t)]I[D_{fs}^{FP}(t)]B[D_{fs}^{BP}(t)]D_{fs}^{EP}(t)$$

where operations *I* and *B* denote binarization by a threshold level T_h. This threshold level should be set between the noise level and the signal level. Operations *I* and *B* are dependent on the following rule:

$$I[D] = 0 \text{ at } D > T_h$$
$$I[D] = 1 \text{ at } D \le T_h$$
$$B[D] = 1 \text{ at } D > T_h$$
$$B[D] = 0 \text{ at } D \le T_h\,.$$

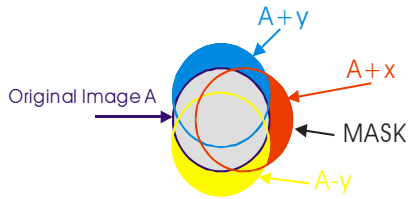
Each mask consists of three adjacent shifted images, which can be implemented easily using address shifting of three adjacent line buffers.

Therefore, the un-normalized moving vector should be modified with the above masking operation as follows:

$$m'_1 = \frac{1}{2}\{D'^{DP}_{fs}(t) + D'^{FP}_{fs}(t)\}$$

$$m'_2 = \frac{1}{2}\{D'^{HP}_{fs}(t) + D'^{BP}_{fs}(t)\}$$

An example of masking to a direction is shown.



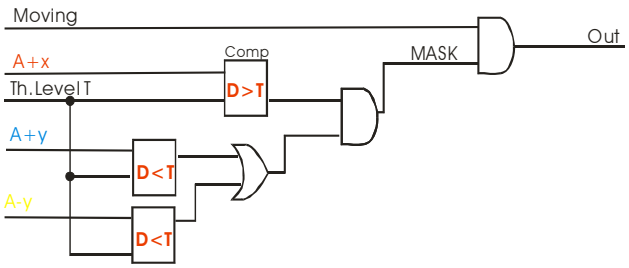


Fig. 4 Schematic of masking operation

3.3 Magnitude Normalization

As the un-normalized moving vector is derived from subtraction between the three adjacent shifted images as described above, the behavior of the vector's magnitude is far from linear to the behavior of the actual moving. The magnitude of the masked (un-normalized) vector does unfortunately create other nonlinearities in that the magnitude of the vector has a local maximum for movement in horizontal and vertical directions. We overcame this problem by rewriting the vector separating its magnitude from the direction. The un-normalized vector should be used to derive only the direction of movement of the ideal moving vector. We found that the magnitude of the ideal vector could be derived from the subtraction image of the un-masked images mentioned above. The number of pixels of the subtracted images that exceed some threshold level is proportional to the magnitude of the actual movement. To discriminate the dependency of the number to the various scenes, we divided the number by the number of pixels of the edge, because the edge is a dominant feature in the subtraction image. The edge image was obtained easily by calculating the average image of the shifted images as follows. (Note that only the fast image $Y_f(x,y,t)$ is used in this operation). In addition, the usual subtraction image between the shifted and the non-shifted images tends to lead to local maxima when the magnitude of movement is the same as the shifting magnitude. However, our subtraction does not tend to have such local maxima because the shifted and the non-shifted images consist of several images accumulated in the time domain.

That is, the number of pixels of $D^{EP}_{fs}(t)$ exceeding the noise level is $d^{EP}_{fs}(t)$ (, where counting is valid except in the unmoving area that is the body of the flying object). This number $d^{EP}_{fs}(t)$ represents the amplitude of movement linearly rather than the un-normalized vector. Next this number should be normalized by the number of pixels of the edge exceeding some noise level. Finally, the moving vector $\vec{M}(t)$ is expressed as follows:

$$\vec{M}(t) = \frac{d^{EP}_{fs}(t)}{e^{EP}_{ff}(t)}(\cos\Theta, \sin\Theta)$$

where $e^{EP}_{ff}(t)$ is the number of pixels exceeding the noise level of an edge image $E^{EP}_{ff}(t)$. This equation indicates the normalized moving vector expressed by two segments of the magnitude and the angle.

The edge image is obtained simply realizing the equations below:

$$E_{ff}(t) = \frac{1}{4}\{D^{HE}_{ff}(t) + D^{DE}_{ff}(t) + D^{FE}_{ff}(t) + D^{BE}_{ff}(t)\}$$

$$D^{HE}_{ff}(t) = |Y_f(x,y-1,t) - Y_f(x,y,t)|$$

$$D^{DE}_{ff}(t) = |Y_f(x-1,y,t) - Y_f(x,y,t)|$$

$$D^{FE}_{ff}(t) = |Y_f(x+1,y,t) - Y_f(x,y,t)|$$

$$D^{BE}_{ff}(t) = |Y_f(x,y+1,t) - Y_f(x,y,t)|$$

It should be remembered that the original image $Y_f(x,y,t)$ is derived from the *recursive integration* method described above. So, the magnitude $d^{EP}_{fs}(t)$ and $e^{EP}_{ff}(t)$ contribute the linearity to the moving representation in comparison with the case using raw images. The angle Θ should be derived from the above un-normalized vectors as

$$\Theta = \tan^{-1}(m'_2 / m'_1).$$

We implemented these trigonometric functions of ***sin, cos***, and ***arctan*** as initial values of memory inside FPGA at 512 steps each.

3.4 Horizontal Camera Integration

We have derived the three normalized moving vectors $\vec{M}_i = (M_{ix}, M_{iy})$ for each horizontal image sensor. However, these vectors still include some ambiguities caused by distortion of the images, variation of distances between the flying object and the observed objects, *etc.*

We attempted to reduce these ambiguities by integrating the three moving vectors. Analytically, the following method is not sufficient to derive the precise attitude of the flying object, but is sufficient to control the flying object in the air for hovering. The following is an attempt to integrate these vectors, which allows derivation of the derivatives of ***Roll***, ***Pitch***, ***Yaw***, and ***Height*** of the flying object.

$$d(Roll)/dt = \frac{1}{2}\{M_{1y} - M_{2y}\}$$

$$d(Pitch)/dt = -M_{0y}$$

$$d(Yaw)/dt = \frac{1}{2}\{M_{1x} + M_{2x}\}$$

$$d(Height)/dt = \frac{1}{2}\{M_{1y} + M_{2y}\}$$

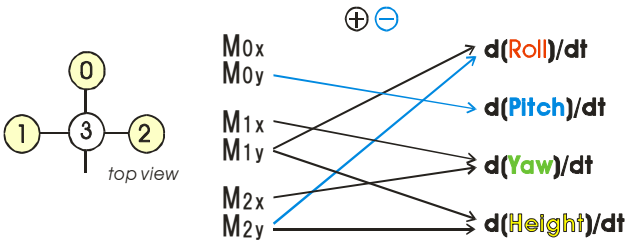


Fig. 5 Schematic of combination logic to derive the attitude of the flying object from horizontal view moving vectors

4. Target Detection

The attitudes derived from the optical flow are derivatives, which means that they are relative values to an experimental field. In addition, we obtain absolute values by viewing two objects on the ceiling. These objects are two lights, separated by a distance of 210 mm and located at a height of 1000 mm between the hovering point of the flying object and the ceiling. We can calculate ***Roll***, ***Pitch***, ***Yaw***, and ***Height*** using the locations of the two images obtained from the other sensor mounted on top of the flying object.

First, we binarized the images obtained by the top camera using an infrared filter and the appropriate threshold level. We implemented a function to calculate the center of attended pixels within some window of an image. A schematic of the function, which consists of a line average module and an address counter, is shown below.

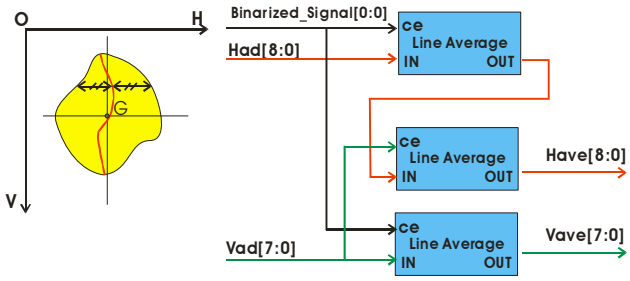


Fig. 6 Schematic of the module for finding the center of the attended area

Thus, the locations of the attended objects can be detected as $\vec{A}(a_1, a_2)$ and $\vec{B}(b_1, b_2)$. Absolute attitudes ***Roll_a, Pitch_a, Yaw_a*** and ***Height_a*** are:

$$\vec{O} = \frac{1}{2}\{\vec{B} + \vec{A}\} = (Roll\_a, Pitch\_a)$$

$$\vec{M}' = \vec{B} - \vec{A} = (M'_1, M'_2)$$

$$Yaw\_a = \tan^{-1}(M'_2 / M'_1)$$

$$|\vec{M}'|^{-1} = Height\_a$$

These equations were implemented using simple operations, such as add, subtract, divide, and arctan.

5. Flight Control Method

For the flight control method, we used a modified PID control method. The following calculations should be performed by the CPU using an appropriate language, but we have fully implemented this method using RTL level design in the FPGA. We planned to use some other CPU, such as the Hitachi SH4, but the weight limitations did not allow this. In the near future, we plan to introduce a software CPU into the FPGA. The calculations that we have implemented in the chip are as follows.

In the case of Roll control, the controlled variable c_r is expressed as:

$$c_r = k_{rp}P_r(r_{rp}, l_{rp}) + k_{rd}D_r(r_{rd}, l_{rd}) + k_{r0}.$$

k_{rp} and k_{rd} are constants, and represent proportional and derivative gains, respectively. r_{rp} is ***Roll_a***, which is derived from the top view, and r_{rd} is ***Roll*** derived from the horizontal view. Function P and D are

$$P_r(r_{rp}, l_{rp}) = r_{rp} \text{ during } r_{rp} \le l_{rp}$$

$$P_r(r_{rp}, l_{rp}) = l_{rp} \text{ during } r_{rp} > l_{rp}$$

where l_{rp} and l_{rd} are limitation constants. k_{r0} is the initial offset constant. In a similar manner, the other controlled variables of ***Pitch, Yaw***, and ***Height*** are as follows:

$$c_p = k_{pp}P_p(r_{pp}, l_{pp}) + k_{pd}D_p(r_{pd}, l_{pd}) + k_{p0}$$
$$c_y = k_{yp}P_y(r_{yp}, l_{yp}) + k_{yd}D_y(r_{yd}, l_{yd}) + k_{y0}$$
$$c_h = k_{hp}P_h(r_{hp}, l_{hp}) + k_{hd}D_h(r_{hd}, l_{hd}) + k_{h0}$$

Thus, 28 controlling parameters are used and tuned to make the object hover.

6. Motor Control

The control variable should be transformed according to the following equations:

$$\begin{bmatrix} Mot0 \\ Mot1 \\ Mot2 \\ Mot3 \end{bmatrix} = \begin{bmatrix} 0 & -1 & -1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & -1 & 1 \\ -1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} c_r \\ c_p \\ c_y \\ c_h \end{bmatrix}$$

where ***Mot0***, ***Mot1***, ***Mot2***, and ***Mot3*** are control variables for each motor of front, left, back, and right, respectively. These control variables are finally output using pulse width modulation at a base frequency of 1 kHz.

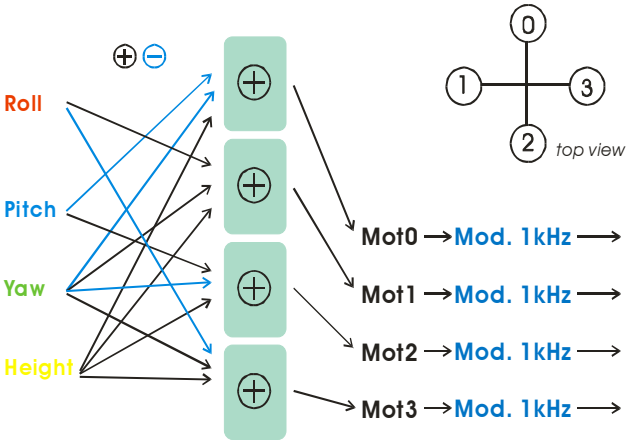


Fig.7 Schematic of transformation from control parameters of Roll, Pitch, Yaw, and Height to each motor output

8. Results and Conclusion

In this study, we attempted to make a flying object stable in the air using a high-speed visual processing controller consisting of FPGA. We implemented a real-time optical flow system with three CMOS sensors and a target detection system with a CMOS sensor at a frame rate of 25 *msec*. We overcame the aliasing problem of the skew line in optical flow processing, and enhanced the signal-to-noise ratio. Finally, we detected the optical flow and target deriving the attitude of the flying object and its location. Using these parameter and other control variables, we controlled roll, pitch, yaw, and height of the object, making it stable in the air for several minutes until the motors began to be overheated.

As the next step, we are planning to implement another method to obtain the locations of the object absolutely using natural visual input. This will be the first study to control a flying object navigating using only natural visual input. Photos of the field experiment are shown below.

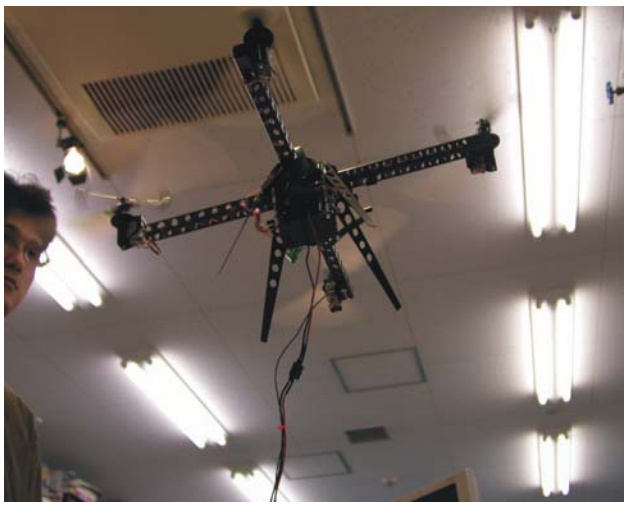


Fig.8 Photo of the flying object

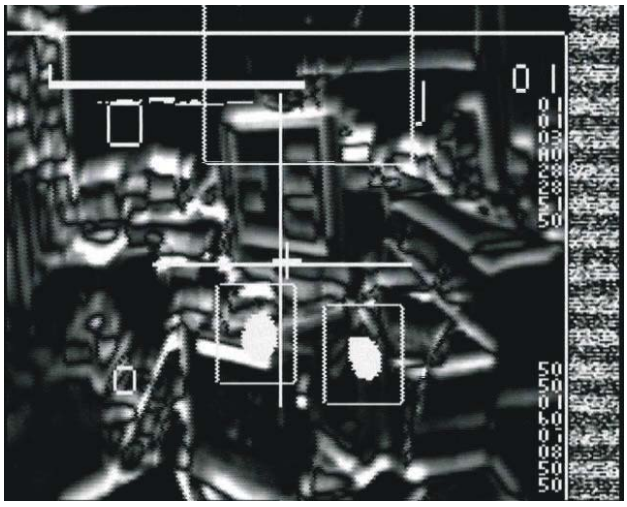


Fig.9 Photo of debugging monitor

10. References

[1] Engineering applications of small brains: N. Franceschini, FED Journal Vol.7 Suppl.2 (1996)

[2] A flying robot controlled by a biologically inspired vision system: M. Ichikawa, H. Yamada, and J. Takeuchi, Proc. ICONIP 2001, p677-680

[3] A flying object using hardware implemented vision processing and motor control system with adaptive network: H. Yamada, J. Takeuchi, G. Matsumoto, and M. Ichikawa, Proc ICONIP 2002, Singapore