Eyebot image processing primitives

Petter Reinholdtsen <pere@td.org.uit.no>

2000-04-21

Abstract

This document covers image processing functions available in Eyebot RoBIOS version 2.3 internal patch level g. It tries to give the reader a good start to use the available primitives as effective as possible. It is based on the RoBIOS source code, to make sure the function descriptions are as accurate as possible.

1 Introduction

Eyebot image functions handle 4-bit gray scale images and 24 bit color images. There are two defined image types, available when including "eyebot.h":

```
#define imagecolumns 82
#define imagerows 62
BYTE image[imagerows][imagecolumns];       /*  5084 bytes */
BYTE colimage[imagerows][imagecolumns][3]; /* 15252 bytes */
```

The image dimensions are 80 rows and 60 columns with one border row on all sides. The color images are RGB images with red as byte 0, green as byte 1 and blue as byte 2.

The memory layout of the image types makes this the fastest way to loop thru the image components:

```
image img;
int row, column;
for (row = 1; row < imagerows-1; row++)
  for (column = 1; column < imagecolumns-1; column++)
    process_pixel(img[row][column]);
```

Note that pixels coordinates go from 1 to imagerows-2 and imagecolumns-2 inclusive to skip the border pixels.

This loop might sometimes be faster, but it process the left and right border pixels as well.

```
image img;
int pos;
for (pos = imagecolumns; pos < imagecolumns*(imagerows-1); pos++)
    process_pixel(img[0][pos]);
```


The RoBIOS kernel has a number of image processing functions available. I will here group them into camera, color, grey-scale and LCD display functions. They are part of the Camera (CAM), Image Processing (IP) and LCD module

In addition to the kernel functions, the libimprov library provides a number of image operations.

2 Camera functions

```
#include "eyebot.h"
int CAMInit (int zoom);
int CAMGetColFrame (colimage * buf, int convert);
int CAMGetFrame (image * buf);
int CAMMode (int mode);
int CAMGet (int *bright, int *off, int *cont);
int CAMSet (int bright, int off, int cont);
```

Eyebot/RoBIOS currently supports three different cameras; Greyscale QuickCam, Color QuickCam and EyeCam (color). The first Eyebots used Greyscale QuickCams, while later models used Color QuickCams. The latest model uses a locally designed EyeCam, as the older QuickCams are no longer produced, and the specifications for the newer QuickCams are impossible to get from Logitec.

Both QuickCam and EyeCam use the same lenses, and these lenses can be replaced to change viewing-angle. Currently we have 33, 41, 43 and 47 degree lenses. When changing lens on the camera, make sure to get the camera back to focus. Focusing is done by moving the lens back or forward by turning it in the socket. The width of the EyeCam base is 1/6 inch.

The following code will grab one image from the camera:

```
#include "eyebot.h"
int camversion, resval, resval2;
image img;
colimage colimg;

while (INITERROR == (camversion = CAMInit(NORMAL)))
  OSWait(CAMWAIT);

if (NOCAM == camversion)
   LCDPutString("No camera");
else if (BWCAM <= camversion && camversion < COLCAM)
   resval = CAMGetFrame(&img);
else if (COLCAM <= camversion && camversion < 0x20)
   resval = CAMGetColFrame(&colimg, 0);          /* color image */
   resval2 = CAMGetColFrame((colimage*)&img, 1); /* grey image */

if (0 != resval)
   LCDPutString("CAMGet{Col}Frame() failed");
```

CAMInit() accepts parameter WIDE, NORMAL or TELE, to change the zoom of the camera. This parameter is used by grey-scale QuickCam only. CAMInit() might fail the first times, and the while loop make sure the camera is initialized anyway. The version numbers returned from CAMInit() are:

0	Greyscale QuickCam
16	Color QuickCam
17	EyeCam

CAMGetFrame() returns a 80x60 4-bit grey-scale image. The bits occupy the lower 4 bit of an 8-bit BYTE.

CAMGetColFrame() returns a 24-bit color image if the second parameter is 0, and a 4-bit grey-scale image if the second parameter is 1.

Each grab function returns as soon as the next complete image is ready. The QuickCam grey scale delivers 20 frames per second. The QuickCam color frame rate is normally 6.5 frames per second. The EyeCam is stable on 3.7 frames per second, using 270000 microseconds to grab one frame.

Figure 1: Two sample 24bit color images taken from the same position. Left is from QuickCam with wide lens, Right is from EyeCam with narrow lens. Images are 3x the original size.

CAMGet(), CAMSet() and CAMMode() can be used to change the current camera settings. CAMMode() is used to enable or disable autobrightness in the camera. Disabling autobrightness currently only works with QuickCam. CAMGet() and CAMSet() is used to change the brightness and color/grey adjustment parameters. They can currently only be used with the QuickCam. The CAMInit() parameter is only used on grey scale QuickCam.

3 Color image functions

Color images are 82x62 including pixel border. The pixels are 24-bit RGB (3 x 8 bit), as fetched from the color camera.

3.1 IPColor2Grey

```
#include "eyebot.h"
int IPColor2Grey (colimage *src, image *dst);
```

This function converts a color image to 4-bit grey-scale images. Pixels are converted using the following approximation (division by 4 is faster then division by 3): grey level = (R+2*G+B)/4.

Figure 2: Converted from color to grey-scale.

There is currently no way this function might fail, so it should always return 0. The function uses 9200 microseconds on the Eyebot and 350 microseconds in the simulator.

4 Grey-scale image functions

Grey-scale images are 82x62 including pixel border and 4-bit grey level values. The lower 4 bit of the pixel BYTE are used.

4.1 IPDither

Grey level	0-3	4-6	7-9	10-12	13-15
Pattern	00 00	00 10	01 10	01 11	11 11

```
#include "eyebot.h"
int IPDither (image *src, image *dst);
```

Converts every second grey-scale pixel and every second row to a 2x2 black and white pattern. It starts in upper left corner (*src)[1][1] and writes the corresponding pattern to (*dst)[1-2][1-2]. The patterns are given in the table to the right. The border pixels are not touched.

Figure 3: Converted from grey-scale to 2x2 dithered.

There is currently no way this function might fail, so it should always return 0. The function uses 4400 microseconds on the Eyebot and 0.31 microsecond in the simulator.

4.2 IPLaplace

```
#include "eyebot.h"
int IPLaplace (image *src, image *dst);
```

Edge detection using the Laplace operator on each pixel using a 3x3 pixel mask.

$$result = \mathrm{abs}(\begin{matrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{matrix}) \tag{1}$$

There is currently no way this function might fail, so it should always return 0. This function uses 21000 microseconds on the Eyebot and 0.11 microsecond in the simulator.

Figure 4: Result of Laplace operator.

4.3 IPSobel

```
#include "eyebot.h"
int IPSobel (image *src, image *dst);
```

Edge detection using the Sobel operator on each pixel using a 3x3 pixel mask.

$$result = \frac{\mathrm{abs}(\begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}) + \mathrm{abs}(\begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix})}{3} \tag{2}$$

Figure 5: Result of Sobel operator.

There is currently no way this function might fail, so it should always return 0. This function uses 35000 microseconds on the Eyebot and 0.42 microsecond in the simulator.

4.4 IPDiffer

```
#include "eyebot.h"
int IPDiffer (image *source1, image *source2, image *destination);
```

Calculate the grey level difference and store result in destination. The following formula is used for each pixel:

$$destination = \mathrm{abs}(source1 - source2) \tag{3}$$

Figure 6: The result of 'Sobel' minus 'Laplace' operator.

There is currently no way this function might fail, so it should always return 0. This function uses 8200 microseconds on the Eyebot and 0.11 microsecond in the simulator.

5 LCD image display functions

The LCD can display a black and white image with geometry 128x64 or convert a 82x62 grey-scale image (excluding the border pixels) to black and white before displaying it in the upper left corner of the LCD display.

5.1 LCDPutImage

```
#include "eyebot.h"
int LCDPutImage (BYTE bwimg[(128/8)*64]);
```

Write a 128x64 black/white image on the LCD display. Each byte is 8 pixel values. The bytes are printed left to right, top to botton. The bits in each byte are printed in right to left order. I.e the bits of the first two bytes in the array are printed in this order: 76543210 76543210.

There is currently no way this function might fail, so it should always return 0. This function uses 5500 microseconds on the Eyebot and 20000 microsecond in the simulator.

5.2 LCDPutGraphic

```
#include "eyebot.h"
int LCDPutGraphic (image *img);
```

Converts and writes a 4-bit gray scale image in the upper left corner of the black and white LCD display. The border pixels are not written. If the grey level is 0-7, the LCD pixel is inverted. If it is 8-15, the pixel is left as it is. If the LCD is cleared before the image is displayed, this will give a simple printout of the image.

There is currently no way this function might fail, so it should always return 0. This function uses 13000 microseconds on the Eyebot and 20000 microsecond in the simulator.

6 Image processing speed

Timings are done on 35 MHz Eyebot Mk3 using RoBIOS v2.3g. The simulator ran on Pentium 133 MHz MMX. ms is time in microseconds, ips is iterations per second when run in a loop.

Function name	Simulator ms	ips	Eyebot ms	ips
CAMGetColFrame()	n/a	n/a	270000	3.7
IPColor2Grey()	350	2900	9200	110
IPDither()	0.31	3200000	4400	230
IPLaplace()	0.11	9100000	21000	49
IPSobel()	0.42	2400000	35000	28
IPDiffer()	0.11	9100000	8200	120
LCDPutImage()	20000	50	5500	180
LCDPutGraphic()	35000	29	13000	76

7 References

Eyebot RoBIOS library documentation, T. Bräunl, K. Schmitt, T Lampart 1998.
http://www.ee.uwa.edu.au/ braunl/eyebot/ftp/ROBIOS/library.html

Eyesim - Eyebot simulator, N. Tay, E. Nichols, G. Ong 1999.
http://www.ee.uwa.edu.au/ braunl/eyebot/sim/sim.html

Digital Image Processing, R. C. Gonzales and R. E. Woods, Addison-Wesley 1992

8 Changelog

2000-04-21 Converted base document from HTML to TeX. Correct name of EyeCam from Eyebot.

2000-01-17 Removed some spelling errors and corrected the size of the eyecam base.

2000-01-10 First version released to the public.