GEU111 – High-Tech Tools and Toys Lab
Lab 8: Identifying Colors from a Video Cam Display

In this lab we will learn how to download an image from video cam and analyze the output to determine the color of the object that the video cam is focused on. This is the first step in creating a system that sorts ping-pong balls by color.

A. Downloading a Video Cam Image

We have provided two functions "`void image.Capture(void)`" and "`void image.Get_Pixels(long int image_array)`" in source file `Binarize.cpp` referenced in the header file `Binarize.h` you can use in this task. The C++ line:

```
image.Capture();                    // captures image in file
```

causes the video cam to capture an image to a JPG file in your project directory.

The C++ line:

```
image.Get_Pixels(image_array);      // downloads image
```

accesses the image that was most recently "Captured" and loads it into a long integer array `image_array`.

To use these functions you will need to add `Binarize.cpp` to your project as a "Source File" and include `Binarize.h` as a "Header File." You will find these existing files in the directory `C:\Program Files\National Instruments\NI-DAQ\Include`. Use the entire path in your "`#include`" statement unless you have set a path to the directory above.

At the top of your program you will need to declare the array variable `image_array` as a `long int` and declare `image` as a new variable type ("class") of type "Binarize" by the following commands:[1]

```
long int image_array[76800];
Binarize image;
```

The images that we will download is an array of values that represent each pixel of a 320 x 240 image (320 horizontal x 240 vertical pixels=76800 total pixels). The value for each pixel is a 24-bit integer of which the lowest 8 bits (bits 0-8) represent the "Blue" value of the pixel, the next 8

---

[1] Note that this process is similar to the way we read from a file using the `<fstream>` library:

```
double x;                       // defines data variable "x" as double
ifstream infile;          // defines "infile" as a file class
infile.open("data.dat");        // opens the file "data.dat" as "infile"
infile >> x;              // reads from "infile" into variable "x"
```

bits (9-16) represent the "Green" value of the pixel, and the highest 8 bits (17-24) represent the "Red" value of the pixel. This encoding scheme is know as RGB, where the intensity of the red, green, and blue color of the pixel can each be represented by a number from 0 to 255 (=2^8-1).

B. Creating a Two-Dimensional Image Matrix

The first step that you will need to do is to convert the `long int` array `image_array` into a two-dimensional matrix. The image is 320 pixels wide by 240 pixels high. If we use standard matrix terminology, the matrix will have 240 rows and 320 columns. The data is stored in `image_array` by rows: the first 320 elements represent the pixels in the first row; the next 320 elements are the pixels in the second row, etc. The data is all from left to right, *not* in a raster pattern.

Prelab Task 1: Write a function `void make_matrix_cam_xy(long int Av[], long int Am[][320], int nrows, int ncolumns)` – where xy are your initials – that converts an array `Av[76800]` into a matrix `Am` with 240 rows and 320 columns, assuming that the data is stored in rows in `Av` (the column index changes faster than the row index). Modify and test your function by writing a `main` program that defines an array `Av = {1, 2, 3, 4, 5, 6}` and calls `make_matrix_cam_xy` to convert it into a 2x3 matrix `Am` such that `Am[1][0]` = 4, for example.

C. Extracting the [R,G,B] Values

Each element of the array `image_array` or of your matrix `Am` is a long integer which includes the R, G, and B values as numbers between 0 and 255 in 8-bit segments of the integer as explained above. Derive an algorithm using bitwise operators to extract out the R, G, and B values from a long integer, and convert them to ratios *r*, *g*, and *b* with values between 0 and 1 ($r, g, b \in [0,1]$).

Prelab Task 2: Implement your algorithm in a function:
`void RGB_xy(long int A, double& r, double& g, double& b)`
where "`xy`" are your initials. Test your program with A=1 (*r*=0, *g*=0, *b*=1/255), A=256 (*r*=0, *g*=1/255, *b*=0) and A=65536 (*r*=1/255, *g*=0, *b*=0). What value do you get for *r*, *g*, and *b* for A=11367205? What values would you expect to get? (*Hint:* The MATLAB functions "dec2hex.m" and "hex2dec.m" might help to find the answer to this question.) Why do you only need to have values for red, green, and blue to make up a full-color image?

D. Identifying Color Hue: RGB to HSV Conversion

It can be difficult to identify a red, yellow, or white object by setting up tests on the RGB values because the intensity of the R value, for example, can depend not only on how red the object is, but also on the intensity of the illumination. Clearly, you would want to look at some combination of the ratios of the RGB values to determine color.

This problem has been addressed by the HSV (Hue-Saturation-Value) color coding scheme. The Wikipedia entry on HSL and HSV (http://en.wikipedia.org/wiki/HSL_and_HSV) has a good presentation of the RGB, HSV, and HSL color schemes (with colors!) For our purposes in this lab, we will present only the equations of the RGB to HSV conversion. If the R, G, and B components of color pixel are converted to values between 0 and 1, $r, g, b \in [0,1]$ and *max* is the greatest of *r*, *g*, and *b*, and *min* in the least, then the hue angle is:

$$h = \begin{cases} 0^\circ, & \text{if } \max = \min \\ 60^\circ \times \dfrac{g-b}{\max - \min} + 0^\circ, & \text{if } \max = r \text{ and } g \geq b \\ 60^\circ \times \dfrac{g-b}{\max - \min} + 360^\circ, & \text{if } \max = r \text{ and } g < b \\ 60^\circ \times \dfrac{b-r}{\max - \min} + 120^\circ, & \text{if } \max = g \\ 60^\circ \times \dfrac{r-g}{\max - \min} + 240^\circ, & \text{if } \max = b \end{cases}$$

The saturation *s* is: $s = \begin{cases} 0, & \text{if } \max = 0 \\ \dfrac{\max - \min}{\max}, & \text{if } \max \neq 0 \end{cases}$ while the value *v* = max.

Prelab Task 3: Implement the RGB to HSV algorithm in a function: `void RBG_to_HSV_xy(double r, double g, double b, double& h, double& s, double& v)`. Test your function with *r*=0.5, *g*=1.0, *b*=0.5 which should return an HSV result (120^o, 0.5, 1.0).

By looking at the hue color wheel in Wikipedia decide what hue angle range should be called "red," which hue angles should be called "green," and which should be called "blue." You will need this criterion to distinguish different sheets of paper in this lab, and to determine the color of ping-pong balls in your sorting mechanism. How would you identify "white"?

E. Laboratory Procedures

You will be provided with a video cam and several sheets of colored paper. You are to write a program that will accurately identify the "red" paper, the "blue" paper, the "green" paper and the "white" paper from the video cam image, regardless of the level of illumination (how much light is shining on the paper).

Lab Task 1: Mount the video cam on the ring stand and direct it toward the clip board. Set up a flexible lamp so that it shines on the clipboard. Write a C++ program that prompts the user to enter a positive number and, when you do, calls `image.Capture`. (If you enter a negative number the loop ends and the program ends.) Look in your project file for a .JPG file and click it to view your image. Adjust the position of the video cam and repeat the process until the clipboard fills the image. (Since the `image.Capture` program uses the same file to write

the new image in, you will have to exit the image window after you view it before you can take another image.)

Lab Task 2: Write a program that will:

i. Prompt the user to put a piece of paper in the clipboard
ii. Capture an image from the videocam (using `image.Capture`).
iii. Convert the image to a matrix of 240 rows by 320 columns (using `image.Get_Pixels`)
vi. Extract out the r, g, and b values for each pixel into three 240 x 320 matrices, one for the r, one for the g, and one for the b values of the image (using your `make_matrix_cam_xy` and `RGB_xy` programs).
v. Average the r, g, and b values of all 76800 pixels of the image to get an overall rgb value.
vi. Covert the r, g, v values into a hue angle (using your `RBG_to_HSV_xy` program).
vii. Print out to the console "The sheet is red," "The sheet is green," "The sheet is blue," or "The sheet is white"
viii. Prompt the user to put up another sheet of paper. Determine some way to enable the user to end the program, in which case the console should print: "Thanks for using ColorMaster."

Demonstrate to the instructor or TA that this correctly reports on the color of the paper at low levels of illumination as well as at high levels of illumination (when the flexible light is off and when it is on).

Lab Task 3: Try to create a discriminator that works by creating a threshold on the RGB values directly. Report your observations on how successful it is compared to your hue discriminator system.

Extensions

1. Modify your program so that it only averages the center of your image, ignoring the outer band of n pixels where n can be set from 0 to 120 (vertically) and 0 to 180 (horizontally). Check that this accurately reports the color of the paper when the video cam is moved back so that the field of view contains more than just the clipboard.

2. Modify your program so that it divides the image into quadrants and reports back on the color in the upper left, upper right, lower left, and lower right quadrants of the image. Demonstrate that this correctly reports the colors when two sheets of different colored paper are overlapped.