Group Name: LITTLE WHITE'S FANS ASSOCIATION
Group Members:
Di Wu, Disheng Hou, Fang Shu, Haosen Xing, Qi Wang, Xiaoxiong Li, Zhangsihao Yang
Team A: Di Wu, Qi Wang, Xiaoxiong Li, Zhangsihao Yang
Team B: Disheng Hou, Fang Shu, Haosen Xing

Detailed Design Report

Detailed Function Design

In this part of report, the detailed function design will be introduced, including the target of the functions, the I/O type of the function and the algorithms in the functions.

1.Adjust the Color of Channel and Auto White Balance of the Image

This function is written by **Di Wu**. The temperature function adjusts the color temperature of a given picture. There are two modes for users to choose: auto and manual. The temperature function in auto mode allows users to just click a button and realize Auto White Balance automatically using gray world algorithm. By choosing manual mode, users could specify which color channel to change and how much the intensity needs to be changed in this channel. The function has a form like:

void temperature (Mat& src, int channel, int gain, mode md);

where the arguments are:

1. Mat& src: The picture that needs to be changed.

2. int channel: The channel that needs to be changed if the mode is manual. 0 for b, 1 for g, 2 for r.

3. int gain: The intensity changes of the channel. Range: -255 to 255.

4.mode md: The mode that users could choose which has two values: AUTO and MANUAL.

What the function does in auto mode is:

1.Check if the input channel and md are valid, if not, print out a message and return.

2.Split the given Mat into three channels.

3.Calculate the average value of each channel: $\bar{B},\ \bar{G},\ \bar{R}$.

4.Calculate the gain of each channel using the average value: $\mathrm{KB} = (\bar{B} + \bar{G} + \bar{R})/(3\bar{B})$, $\mathrm{KG} = (\bar{B} + \bar{G} + \bar{R})/(3\bar{G})$, $\mathrm{KB} = (\bar{B} + \bar{G} + \bar{R})/(3\bar{R})$.

5.Change the values of each channel using the gains: $\mathrm{b} = \mathrm{b} * \mathrm{KB},\ \mathrm{g} = \mathrm{g} * \mathrm{KG},\ \mathrm{r} = \mathrm{r} * \mathrm{KR}$.

6.Merge three channels together and return.

What the function does in manual mode is quite like what it does in auto mode, except that after splitting into three channels, it simply adds the gain (provided by users) to the channel (specified by users), and then merges channels together and return.

2.Adjust the Brightness and Contrast of the Image

The function by **Disheng Hou** is to adjust the brightness and contrast of an image, the pseudo codes is of contrast process:

1.Input an image(Mat), and α(double).

2.Use the equation: g(i,j)= αf(i,j)+ β, to process the input(Mat), and α is to adjust the contrast, the β is always 0.

3.Return the processed image(Mat).

The pseudo codes is of brightness process:

1.Input an image(Mat), and β(double).

2.Use the equation: g(i,j)= αf(i,j)+ β, to process the input(Mat), and β is to adjust the contrast, the α is always 1.

3.Return the processed image(Mat).

3.Smoothing and Sharpening the Image

Image Smoothing (Image Blurring)

This function is written by **Haosen Xing**. Convolving the image with a low-pass filter kernel is a useful method to achieve image blurring. What it does is to remove the high frequency content, such as noise and edges, from the image.

There are usually four methods to blur images, here we consider averaging and gaussian blurring.

1.Averaging

Here we will use a normalized box filter, which uses function cv2.blur() or cv2.boxFilter() to average all the pixels under the kernel area and then replace central pixels with the new averages. Meanwhile, we should define the width and height. A 3*3 normalized box filter should be:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Pseudocode:

1.inport the former image

2.define the width and height of image

3.use blur function

4.create windows and show the images

2.Gaussian Blurring

Instead of using a box filter, we use function cv2.GaussianBlur(). We have to specify the width and height of kernel (positive and odd). Meanwhile, we should define sigmaX and sigmaY, which are standard deviations in X and Y direction, respectively. If only one of them is known, the other one should be same.

Pseudocode:

1.inport the former image

2.define the width and height of image

3.use GaussianBlur function

4.create windows and show the images

Reference: https://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html

Image Sharpening

We would like to use unsharp masking.

We smooth the image with Gaussian filter with given size and standard deviation;

We substract the blurred image from the original image to find edges;

Mask(x,y) = Origin(x,y) – Blurred(x,y)

Multiply the mask (edges only) by K to enhance edges regions in order to add weighted portions.

Result(x,y) = Origin(x,y) + K * Mask(x,y)

Pseudocode:

1.get user input K (K>=1)

2.define the width and height of image

3.use GaussianBlur function

4.use subtraction to get mask

5.add a weighted value K to the mask

6.sum with the origin image

7.return the processed image

Reference: https://dsp.stackexchange.com/questions/15176/sharpening-an-image-using-emgu

4.Changing the Saturation of Image

This function is written by **Zhangsihao Yang**. HSV (Hue, Saturation, Lightness) and HSV (Hue, Saturation, Value) are two alternative representations of the RGB color model, designed in the 1970s by computer graphics researchers to more closely align with the way human vision perceives color-making attributes. In these models, colors of each hue are arranged in a radial slice, around a central axis of neutral colors which ranges from black at the bottom to white at the top. The HSV representation models the way paints of different colors mix together, with the saturation dimension resembling various shades of brightly colored paint, and the value dimension resembling the mixture of those paints with varying amounts of black or white paint. The HSV model attempts to resemble more perceptual color models such as NCS or Munsell, placing fully saturated colors around a circle at a lightness value of 1/2, where a lightness value of 0 or 1 is fully black or white, respectively.

In the purpose of changing the saturation, the HSV color space is chosen. So the function named change_colour_saturation is written. In this function, first, convert the color of the image from GRB color space to HSV space. Then change each pixel on HSV color space. And then convert the image in HSV space back to RGB color space. The pseudocode is as below,

void change_colour_saturation(image, h , s , l)

{

Create an image named HSV

Convert the image to HSV color space and save it into HSV

For each pixel in HSV

{

Change the HSV value according to h, s, and, l

}

Convert HSV back to RGB color space

}

So the input is the pointer point to the image the output is void. The function named cvtColor is used to transform the image in RGB color space into HSV color space and also change it back to RGB color space. The code is as following:

```
cvtColor(image, hsv, CV_BGR2HSV);
cvtColor(hsv, image, CV_HSV2BGR);
```

Then change the color in HSV color space according to each pixel. The code is as following:

```
hsv.at<Vec3b>(y, x)[1] =  saturate_cast<uchar>(hsv.at<Vec3b>(y, x)[1] + inc);
```

The indice [1] means the hue is changing. And if changing to [2] means the saturation is changing.

5.Adjusting the Hue of the Image

This function is written by **Fang Shu**. This function is to adjust the hue of the image to change the whole color of the image which can produce fantastic effect. The pseudocode is:

Input Opencv Mat file called cvsrc cloned from globalsrc which contains data for RGB color range.

Use cvtColor BRG2HSV function to change RGB color range to HSV color range.

Use Split function to split the HSV color range data into hue.

Use minMaxIdx function to set minimum and maximum for value of hue.

Initial a variable called hvalue for hue.

Find the hue data for each pixel and use saturate_cast<uchar> function to change the value of hue whether the value is larger than hvalue or smaller than hvalue.

Use cvtColor HSV2BRG function to change HSV color range back to RGB color range.

Output the new Mat file called new_image and assign it as globalsrc.

6.Adjusting Highlight and Shadow part of the Image

This function is written by **Qi Wang**, and uses a linear variation to adjust the greyscale of the image, in this way, we can get an enhanced image. In this kind of image processing transform, each output pixel's value depends on only the corresponding input pixel value (plus, potentially, some globally collected information or parameters). We use the linear function to change the pixel of the image. And the linear function is the most common function.

We use the following equation to adjust the greyscale.

$g(x)=\alpha f(x)+\beta$

f(x) is the source image pixels, and g(x) is the output image pixels. More conveniently, we can write in this way:

$g(i, j)=\alpha \cdot f(i, j)+\beta$

where i and j indicates that the pixel is located in the i-th row and j-th column.

We set α=1. This indicates the contrast is constant. And we change the brightness by changing the value of β. If d is positive, the brightness increases. If β is negative, the brightness decreases.

(If We set α<0, the greyscale will change inversely. The former brighter area will become darker)

In our code, we first find the whole range of the greyscale of the image. Then we set three sub-range of the greyscale (from low to high-range 1, range 2, range 3). Then for all the pixels whose greyscale is in the range 1, they belong to shadow part. For all the pixels whose greyscale is in the range 3, they belong to highlight part. Then we ask the user to input different β to adjust the highlight and shadow.

pseudocode

1.input the former image

2.get the maximum greyscale and minimum greyscale of the whole pixels

3.seperate the range into three sub-ranges equally(range1,range2,range3)

4. if the greyscale of one pixel is in the range1

input β to adjust the greyscale

put the pixel in the linear equation 1(with α=1 and input β)

if the greyscale of one pixel is in the range3

input β to adjust the greyscale

put the pixel in the linear equation 1(with α=1 and β)

5. return the processed Mat class image.

7.Color filters and Stylized Filters

This function is written by **Xiaoxiong Li.** This job is to design various filters including color filters and stylized filters. The color filters include Crema, Gingham, Moon, Slumber and so on. The stylized filters include different types of textures and style to make the picture more beautiful.

Pseudocode:

1.Store the data of the image into Mat class, using imread method.

2.Create function 'filter(Mat src, int classification)' to process the image as required by user. The 'classification' is received by UI. It tells which kind of filter is asked by the user.

3.In 'filter' function, three colors of each pixel are extracted, which are stored in three positions in memory respectively. Function 'getBGR' is called to process the three colors.

4.In 'getBGR', three colors are expressed in char data type, which means they are range from 0 to 255. Each kind of color filter need a specific formula to calculate the proportion of the R, G, B color. Based on the class of the filter user requiring, 'getBGR' choose different equation to process the colors.

5.For stylized filers, we need to call functions in OpenCV library to process the image, such as edgePreservingFilter, detailEnhance and so on.

6.After processing the colors, the processed Image is returned to UI.

Assembly Design

The final platform for assembly is QT Creator, **Disheng Hou, Fang Shu,** and **Haosen Xing** will assemble all the functions and write the code of GUI, and the assembly leader is **Disheng Hou.**

The function we use all return an Mat class in OpenCV, and we use this return value to communicate among all functions. In the mainwindow class, there will be two global variables(Mat) to send input to the sub functions, and collect the output of sub functions.

For GUI designing, we use QT designer class to create the mainwindow and sub-widget and use scroll bar, button or other controls to adjust the image. In each sub-widget class, we use a Mat class variable to receive the input from mainwindow, and send output to the mainwindow, and show the processed image on the sub-widgets.