CSC3002

Introduction to Computer Science: Programming Paradigms

Project Report

A Multifunctional Image Editor: Unipicture

Ran Hu	116010078
Jiayi Qiu	116010182
Shaoyu Wang	117010262
Zhixian Hu	116010082
Xuening Zhang	116010302
Jinjing Yang	115020157

Course Coordinator: Prof. Rui Huang

The Chinese University of Hong Kong, Shenzhen,

Spring 2019

1. Introduction

Our image editor has all the functions as stated in the proposal, and several additional functions -- Big Eye, Undo & Redo etc. It not only has an elegant and friendly interface, but also has many useful functions that enable users to create beautiful and creative pictures.

Following are the design and functions of *Unipicture*:

1.1 GUI Design

The graph below shows our GUI design for this image editor. Users can import images, which will be displayed on the canvas. Functional buttons are arranged on the left.

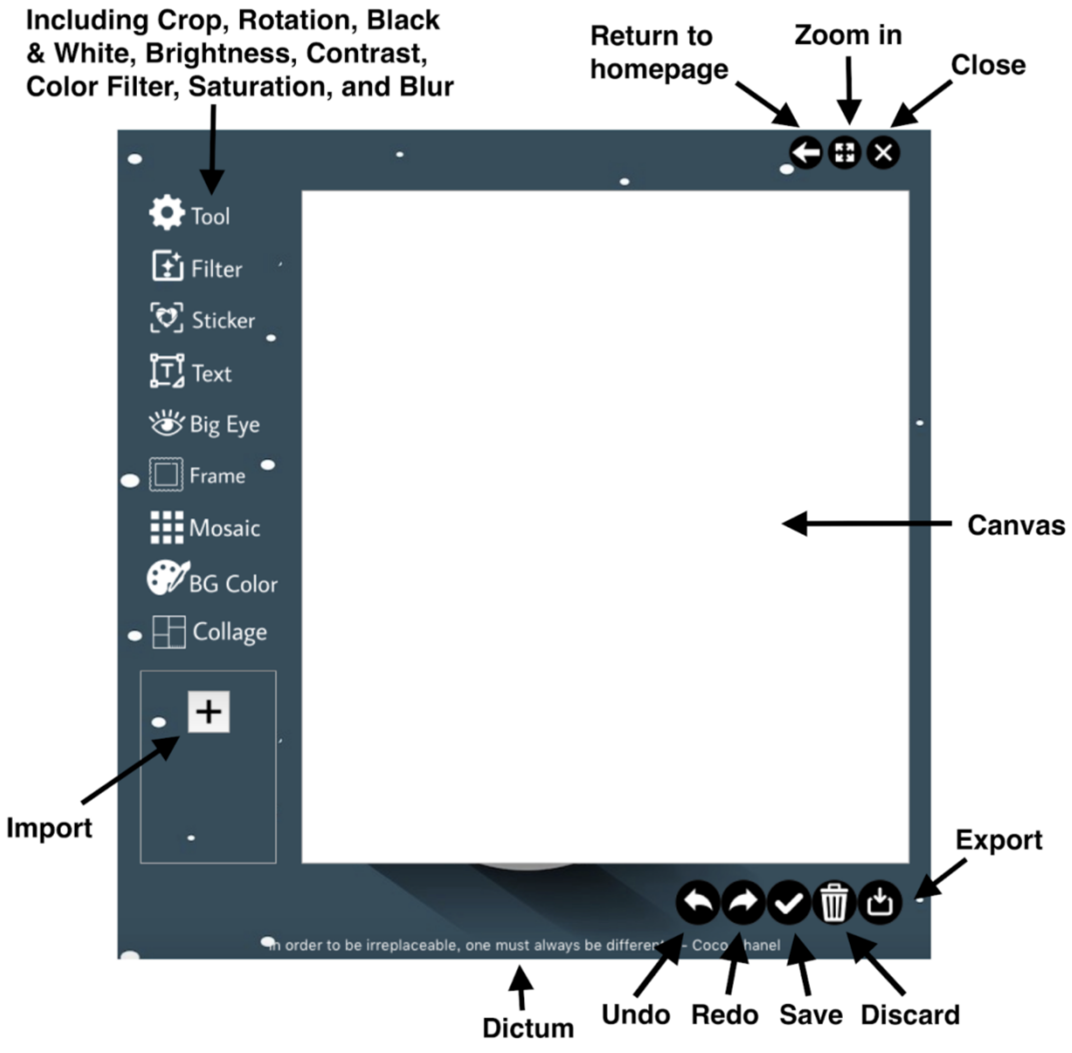

Including Crop, Rotation, Black & White, Brightness, Contrast, Color Filter, Saturation, and Blur
Return to homepage
Zoom in
Close
Tool
Filter
Sticker
Text
Big Eye
Frame
Mosaic
BG Color
Collage
Canvas
Import
Export
In order to be irreplaceable, one must always be different. - Coco Chanel
Dictum
Undo
Redo
Save
Discard

Fig. 1: GUI Design

1.2 Functions

- **Tool:** Basic operations on an image including Crop, Rotation, Black & White, Brightness, Contrast, Color Filter (Fig. 16b), Saturation (Fig. 2b), Blur.

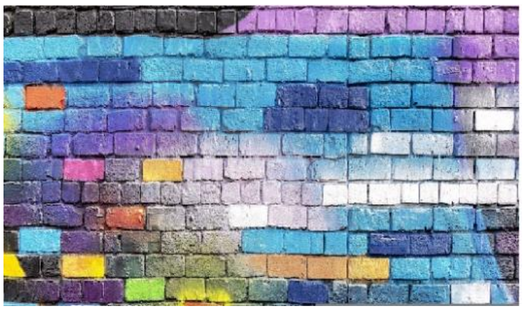


Fig. 2a: Original Image

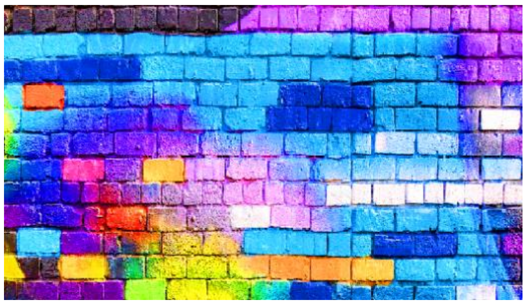


Fig. 2b: Image after Saturation Operation

- **Filter:** Simple filters and advanced filters. The division criterion is the complexity of the filter operation. For simple filter, pixels after applying the filter are simply calculated by using one single pixel in the original image. For advanced filter, each new pixel will be calculated based on pixels nearby or some more complex rules (Fig. 4 Pepper Filter).
- **Sticker:** Allow users to choose one sticker we provide. Click on the image to indicate where they want to add that sticker (Fig. 3).
- **Text:** Add a text on chosen position of an image. User can adjust the size of the text (Fig. 4 Text).

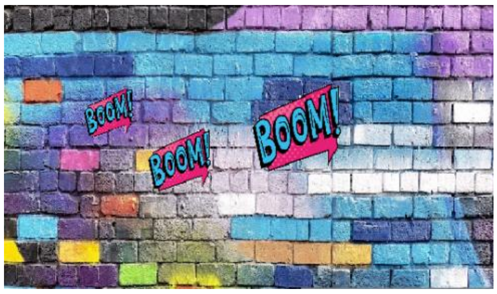


Fig. 3: Image with Stickers

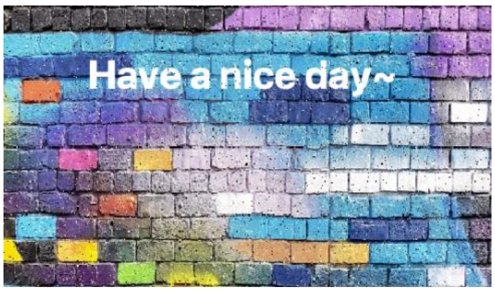


Fig. 4: Image with Text and Pepper Filter

- **Big Eye:** Enlarge the eyes of a person. Users can adjust the enlarged size (Fig. 5).

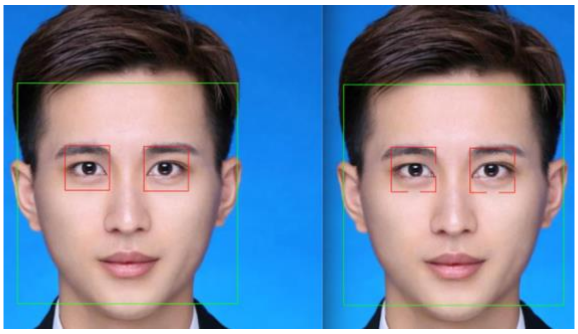


Fig. 5: Origin Image & Eye-enlarged Image

- **Frame:** Add a frame on an image. Users can choose eight types (Fig. 6 Frame).
- **Mosaic:** Blur the chosen area of an image. Users can choose three sizes (Fig. 6 Mosaic).
- **Background Color:** Change the background color of a portrait. Users can choose three colors: red, white, and blue (Fig. 17).
- **Collage:** Splice two or more pictures horizontally or vertically (Fig. 7).

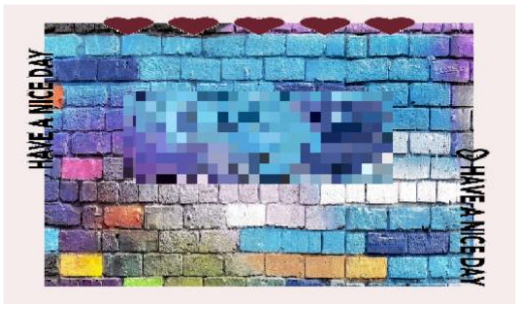


Fig. 6: Image with Frame & Mosaic

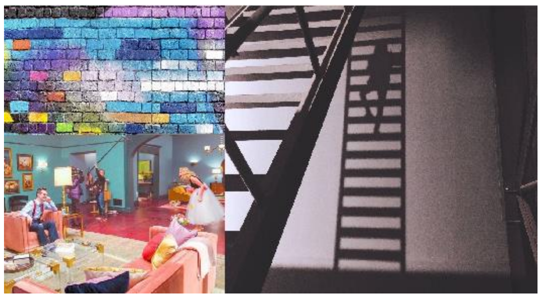


Fig. 7: Three-Images Collage

- **Import:** Allow the user to import as many images as they want, and display them as image buttons in the frame. Click one image button, and the corresponding image will be displayed on the canvas (Fig. 15).
- **Undo & Redo:** Undo can return to the last saved image in the undo stack, and put the undone image to the redo stack. Redo can return to the next undo image.
- **Save & Discard:** Save the current image on canvas into the undo stack. Discard the image can make the canvas empty.
- **Export:** Export the image to a local address on users' computer.

With the help of websites and blogs including Stack Overflow, CSDN, Qt documentation, GitHub, and books including *C++ GUI Programming with Qt 4 (2nd-*

ed), OpenCV 3 Computer Vision Application Programming Cookbook, we achieved all the expected functions.

2. Division of work

Name	Division
Ran Hu	1. Import, Undo & Redo, Save & Discard, and Export functions 2. Mouse controls (Crop, Sticker, Text, and Mosaic) 3. Link GUI with Big Eye, Frame, Background Color, and collage
Jiayi Qiu	1. GUI Interface, signal-slot connections 2. Text function, random dictum display feature 3. Link GUI with Tool functions (Rotation, Black & White, Brightness, Contrast, Color Filter, Saturation, Blur) and Filters
Shaoyu Wang	1. Mosaic function 2. Big Eye function 3. Change Background Color function
Zhixian Hu	1. Tool functions (Crop, Rotation, Black & White, Brightness, Contrast, Color Filter, Saturation, Blur)
Xuening Zhang	1. Sticker function 2. Frame function 3. Collage function
Jinjing Yang	1. Basic Filter functions 2. Advanced Filter functions

3. Difficulties and Solutions

Ran Hu

1. The transformation between data types **cv::Mat** and **Qimage**

There are two troublesome problems. First, the default format of **cv::Mat** is BGR, while that of QImage is RGB. Second, the read-in image is of the form ARGB, which use 4 channels to represent an image, while our algorithms based on Mat type is 8U3C (8 bits and 3 channels RGB).

Solution: use **cvtcolor()** in OpenCV and **convertToFormat()** in QImage to make conversions between BGR and RGB. However, there are still some problems remain because we cannot detect whether an image is in BGR or RGB. Therefore, sometimes it will result in wrongly converting the data type and the channel.

2. The deep copy and shallow copy problem in **cv::Mat** and **QImage**

The copy of one image to another image variable in Mat format is shallow copy. Therefore, those two image variables share the same address. When implementing the functions like Sticker, Filters etc., problems will arise.

Solution: use **Mat.clone()** and **Qimage.copy()** to make a deep copy of an image.

3. The mouse controls and interface interaction

To provide users with a good interface experience, we need to design an elegant mouse control system.

Solution: overwrite the mouse control by creating a class **GraphScreen**, which is a subclass of **QGraphScreen**. We create a new control system using Boolean flag signals. When the mouse movement signals (press, move, release…) are triggered, we exam the flag signals, which are set to be true when calling certain functions.

4. The precise mouse position on the image

Because we can zoom in the main window, and each picture has different size (length and width) on the canvas, it is hard to track the position of the image shown on the canvas and obtain the mouse position.

Solution: we get the mouse position using the **GraphScreen** class mouse event rather than the mouse event in **main_window** class (used to move the whole window). Also, we detect whether the mouse press and release position are outside the image, thus avoiding the crash of the program.

5. The Load function

Load Imported images dynamically into the photo gallery and show each of them on the canvas.

Solution: create a **QList** for **QToolButton** and map the address of the image with its corresponding image button using **setMapping()**. Then, by clicking the image button in the import frame, we can get the image address and display each of them on the canvas.

Jiayi Qiu

1. Generate GUI interface

In order to generate an operation interface for picture editing, one can either use Qt Designer or using coding. For beginners, Qt Designer is good for visualizing but not very convenient to use. It took me some time to understanding how Qt Designer works and what is the difference between using Qt Designer and using coding to generate GUI interface.

Solution: Our image editor has many functions which requires a relatively complex GUI interface, so that using coding is more flexible than using Qt Designer. Only a graphics view, some scroll areas and some layouts were added in a QWidget using Qt Designer. Other components like buttons, sliders, spin boxes as well as signal-slot connections were set by programming.

2. Interface Switching

Our main operation interface includes a canvas and some buttons for functions like basic tools, filters, adding stickers and so on. For each function, it can also require some relative components. For example, some buttons are need to allow users to launch crop, rotation and other operations belong to basic tools function. Therefore, interface switching is needed.

Solution: The solution of adding new Qt Designer **.ui** files were considered. Although it can be more convenient to control components in each interface in this way, it can be inconvenient for users to use and. To make the interface simple and functional, some components in left-side layout and top-most layout are designed to be visible only when the corresponding functions may be launched. When switch to another function which requires its own components, other components will be invisible in these two layouts. Functional buttons belong to different scroll areas. Programme functions were designed to set each scroll area be visible or invisible and these functions will be invoked due to button press signals.

3. <u>Real-time operation with sliders</u>

Some project functions like brightness, contrast, saturation, blur can use sliders to control the degree of picture processing effect. Some functions like big eye can use sliders to control the size of eyes. This requires real-time invocations of functions defined in other files and real-time display of processed images.

Solution: Take brightness function for example, the **"Mat brightness (Mat g_SrcImage, int k)"** subroutine has two parameters. One is the image to be processed in data type Mat, the other one is an integer which is used in calculation to decide the degree of effect. A slider and a QSpinBox are added in the class of main window and have signal-slot connection. Either a value change signal in slider or QSpinBox will call a value change slot in another.

If tool button is released, a slot called **Tool_brightness()** will be called, in which the current image shown in the graphic view will be converted from QImage to Mat and copy to private Mat variable **Temp_main_pic**, set an initial value of QSpinBox and the value changed signal of QSpinBox with a slot called **temp_brightness()**. In this slot function, the brightness subroutine is called with **Temp_main_pic** and the value of the QSpinBox being arguments.

Deep copy **clone()** in OpenCV is used to copy the current image shown in the graphic view to variable **Temp_main_pic,** so that as the slider value changed, the operations are done to the initial image. Otherwise, the operation is irreversible due to repeated addition calculations.

In **Tool_brightness()**, it is necessary to destroy the previous connections of QSpinBox and slider with class slots at first to avoid repetitive connecting, which will resulting reduplicative launches of brightness subroutine if users push brightness button again after they returned from other function interfaces.

The chart of brightness is shown in Fig. 8 below.

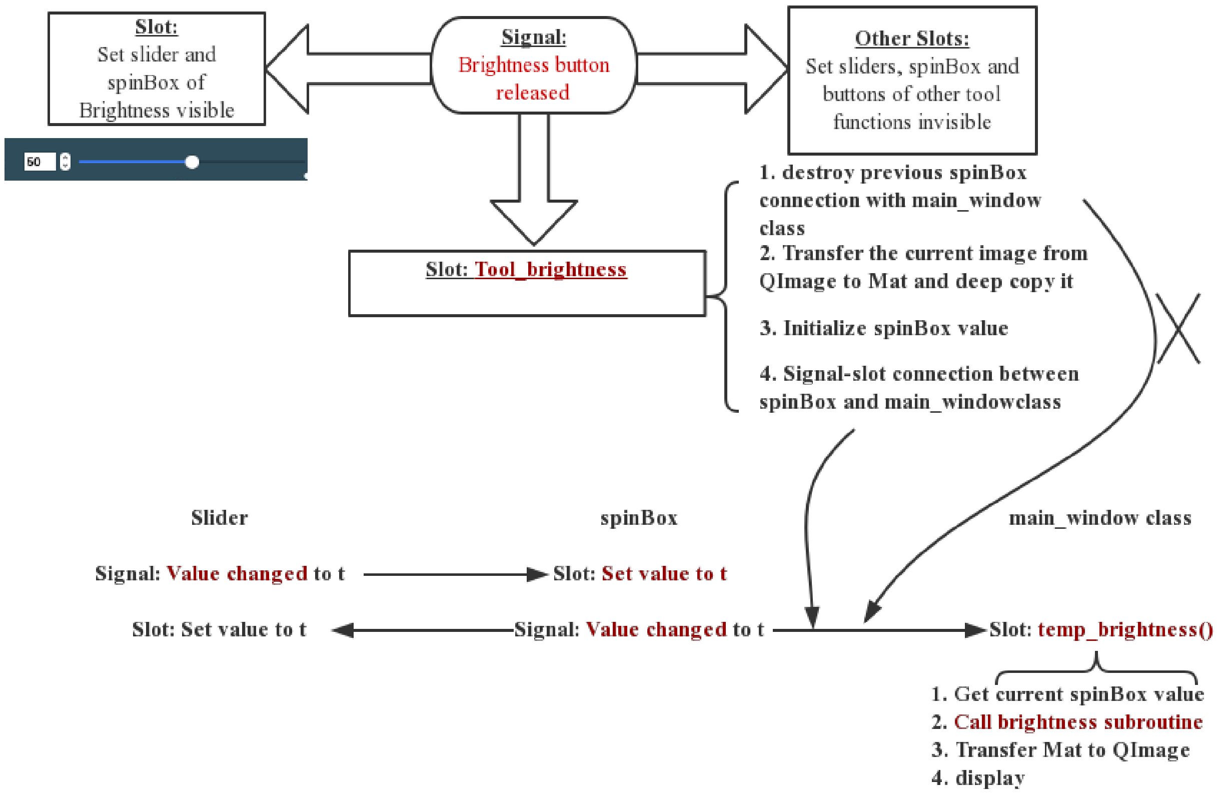

Slot:
Set slider and spinBox of Brightness visible
50
Signal:
Brightness button released
Other Slots:
Set sliders, spinBox and buttons of other tool functions invisible
Slot: Tool_brightness
1. destroy previous spinBox connection with main_window class
2. Transfer the current image from QImage to Mat and deep copy it
3. Initialize spinBox value
4. Signal-slot connection between spinBox and main_windowclass
Slider
spinBox
main_window class
Signal: Value changed to t
Slot: Set value to t
Slot: Set value to t
Signal: Value changed to t
Slot: temp_brightness()
1. Get current spinBox value
2. Call brightness subroutine
3. Transfer Mat to QImage
4. display

Fig. 8: Connections and Programme Flows of Brightness Function

Other functions like contrast, saturation, blur and big eye also use similar programme structure to realize real-time operation with sliders.

4. Add text on images

QGraphicsTextItem class was tried so that users can type text inside the Graphics view. However, this method caused a problem that the image was not well display behind the text.

Solution: A QLineEdit and a QSpinBox is designed to be shown above the canvas after the text button is released. Users can enter text in QLineEdit and give a return to finish their editing. Change the integer in the spinBox can change the text size. Users can press a certain position in the image shown on canvas and the text will appear at that position. The mouse position is obtained by using the **GraphScreen** class mouse event.

In the **GraphScreen** class, there are some public methods and private variables.

public methods:

- **void set_m_text (bool state);** This method is to set or clear text flag, which can determine whether the user is using text function.
- **void add_text (int mx,int my, int size, QString text_content);** This method is to implement text adding operation with the input **QString** text and the input size, at position (mx, my), which can be obtained by mouse event.
- **void set_text_content (QString content);** This method is to set text context.
- **void set_text_size (int size);** This method is to set text size.

private variables:

- **bool m_text = false;** The text flag with initial value false.
- **int text_size = 20;** The text size with initial value 20.
- **QString text_content = “Text";** The tsxt connent with initial string “Text”.

In main_window, private slot **text()** will be launched by the release signal of text button and inside this slot a text flag will set to be true. Fig. 9 below shows how the text-adding function can be implemented.

Other functions which require mouse control and precise mouse position on the image, like Stickers, Crop and Mosaic also used similar flag to invoke corresponding subroutines when the mouse press event happens. These parts were realized by Ran Hu.

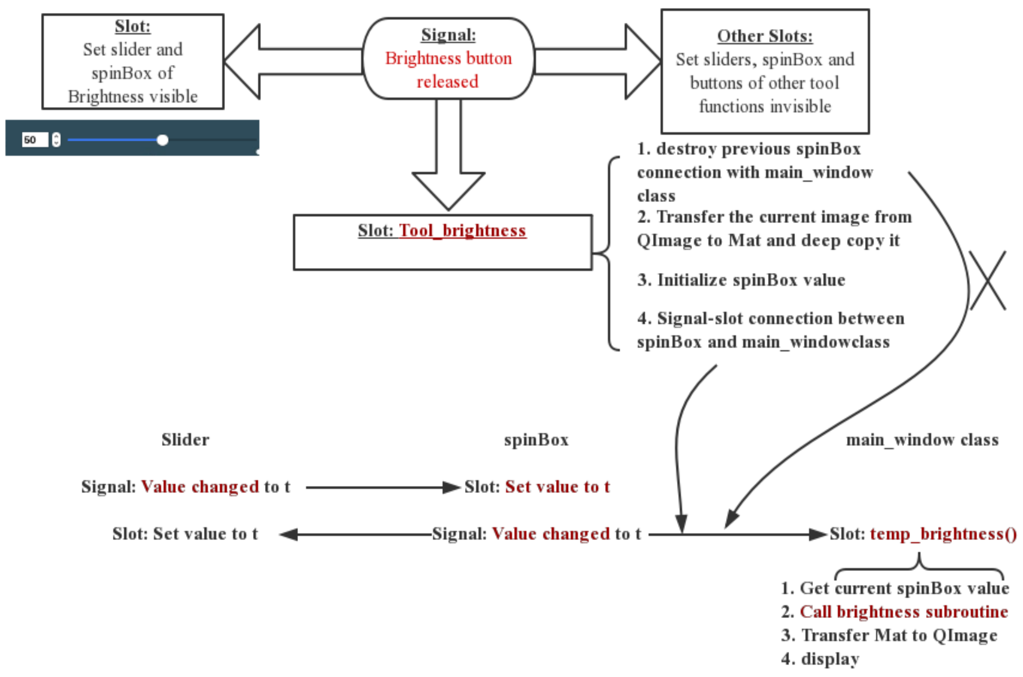

Slot:
Set slider and spinBox of Brightness visible
50
Signal:
Brightness button released
Other Slots:
Set sliders, spinBox and buttons of other tool functions invisible
Slot: Tool_brightness
1. destroy previous spinBox connection with main_window class
2. Transfer the current image from QImage to Mat and deep copy it
3. Initialize spinBox value
4. Signal-slot connection between spinBox and main_windowclass
Slider
spinBox
main_window class
Signal: Value changed to t
Slot: Set value to t
Slot: Set value to t
Signal: Value changed to t
Slot: temp_brightness()
1. Get current spinBox value
2. Call brightness subroutine
3. Transfer Mat to QImage
4. display

Fig. 9: Connections and Programme Flows of Text Function

Shaoyu Wang

1. Mosaic function

This function requires the user to choose two points to determine the diagonal. We need a way to detect which combination the user pick and then determine the square to choose.

Solution: use the coordinate value to determine the left, right, up, down boundary of the square area and then implement the mosaic function on it.

2. Background Change

When implementing this function, the key difficulty is to get the clear and smooth mask information of the identification photo.

Solution: with the help of the OpenCV's haar cascade classifier, the human face can be detected easily. Then set the face phase to be white and the background to be black. After that, use erode and gaussian blur to smooth the boundary.

3. Enlarge the eyes

The hard part is still detecting the eyes and use a proper way to enlarge the eye. The enlarged eyes will look unnatural in the first version.

Solution: The haar cascade classifier will help to detect the eye area. The initial version uses a square area to hold the eye area and enlarge it before sticking it back to the original image. This method will make the resulting image unnatural. In the second version, a circle area is used to hold the eye **Rect** and enlarge it before sticking it back to the initial image, which has much better effect.

Zhixian Hu

1. <u>Mouse response on Crop function</u>

Before GUI part was designed, mouse response was relied on OpenCV for testing. However, program would often crash when implementing mouse response.

Solution: First, I used debug tool to locate the crash point and found that imshow function caused the problem. Then I searched a lot to find solutions and tried them on my computer, but none of them worked. After that, I tested the unchanged program on my partners' computers. The program ran smoothly. The reason behind may be the different versions or settings of OpenCV or computers. This phenomenon may be rare. Thus, this solution is provided just for reference.

Xuening Zhang

1. <u>The Sticker functions</u>

Remove the background of pictures, and only choose the main part we want as a sticker.

Solution: First choose proper pictures. The background of these pictures should be one simple color. The next step is to choose proper parameters for **threshold()** in OpenCV library, which can make the boundary of the sticker clear and smooth. The resulting sticker can be easily attached to the objective image by **Mat.copyTo()**.

Jinjing Yang

1. <u>Complexity and memory</u>

Handle the image with rather low complexity and save memory.

Solution: use call by reference to pass the original image and make a copy of it, preparing for the future operation. Since each filter operation function will return a **cv::Mat** (matrix), at least one copy of the original image should be created. As for time complexity, to handle the whole image pixel by pixel, algorithms with lowest complexity should be chosen. Therefore, two loops are designed for the pixel operation.

2. Simple filter

Get appropriate transformation formula to get new RGB values for the image.

Solution: since the basic filter operations normally adjust the RGB value of one pixel with some pre-defined formulas for color tuning , and most of the pre-defined formulas on the Internet give quite wield result, it costs a long time to modify, test and adjust the parameters for the basic filters.

3. Advanced filter

Use more information rather than the one pixel to make the filters more beautiful, robust and complicated.

Solution: As for advanced filters, the filter operation usually need to consider nearby pixels to adjust current pixel. The information on the Internet for some algorithms can help achieve the goal. For example, the Beauty filter combines Bilateral Filter and Guass Blur to achieve the goal. The diffuse filter use RNG (Random number generator) to pick the RGB value of nearby pixel and use it to replace the value of current pixel. Some filters even provide some customized parameters. For example, the Strike and Volution Filter give user the option to control the intensity of the effect and users can adjust the amplitude and period of the Wave filter. These filters are implemented by controlling the number of loops or changing the center of operation, which is affected by the parameter given by users.

4. Highlights

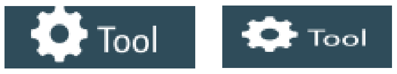
Our project is not only useful and handy, but also creative and interesting, with a number of special features.

4.1 Beautiful GUI:

A splash screen appears waiting for the application to be ready.

Fig. 9: Connections and Programme Flows of Text Function

The buttons on GUI Interface are beautiful and their icons changes when they are pressed. A new class **Button** which is a subclass of **QPushButton** was created to allow button images setting. In Fig. 10 and Fig. 11 are two button samples. The right one in each figure is shown when the button being pressed.



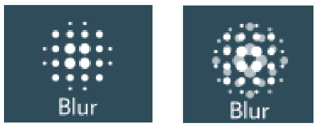
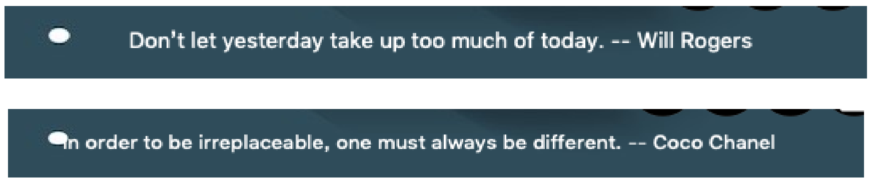


Fig. 10: Button Sample Tool

Fig. 11: Button Sample Blur

The GUI Interface is simple and functional. No other new interface would be generated and interface layout can changed when different function buttons are pushed (Fig. 12 and Fig. 13). It is very convenient for users to operate.

There is a dictum shown on the interface, which is different every time the application opened. This design makes the GUI interface more interesting.



Don't let yesterday take up too much of today. -- Will Rogers

In order to be irreplaceable, one must always be different. -- Coco Chanel

Fig. 14: Different Dictums

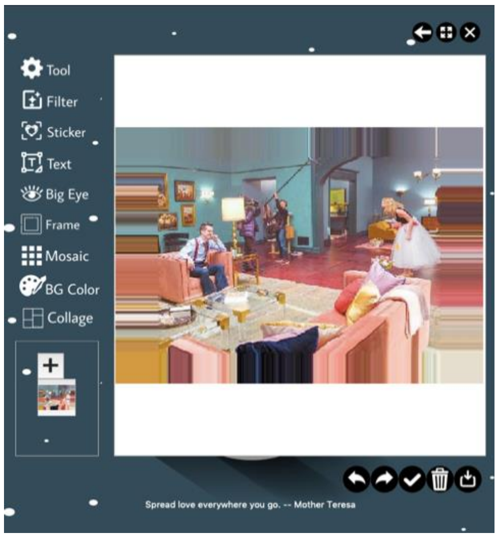


Fig. 12: Initial Interface with image

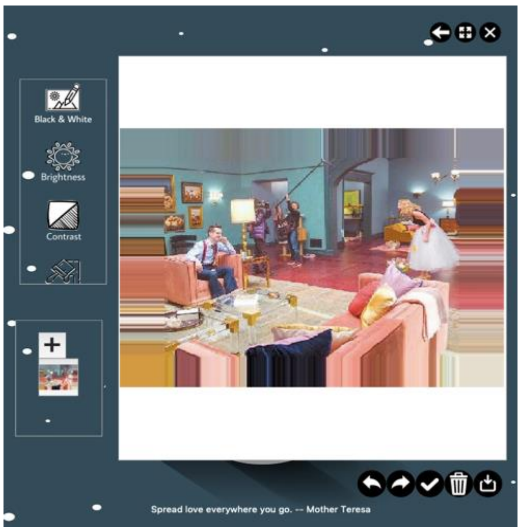


Fig. 13: Interface after Tool Button Pressed

4.2 Import Function:

This function allows users to import as many pictures as they want into the image gallery (Fig. 15). This feature enables users to conveniently refresh the canvas and switch to different images.

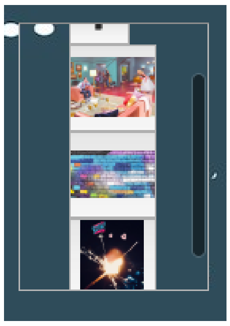


Fig. 15: Image Gallery

4.3 Color filter function:

This function is an extension of the black-white function. Users can select one color A among red, orange, yellow, green, blue (Fig. 16b), purple and pink. The image will be transformed into A-black-white mode (the A color part remains unchanged, while the other part becomes black-white). The division of color is not based on a certain standard of hue, but is designed by ourselves. We tested several standards and found that the results were not satisfying, so we divided color by ourselves. In our design, there is no strict rule between colors, but are some overlaps on neighboring colors to meet a broader command.

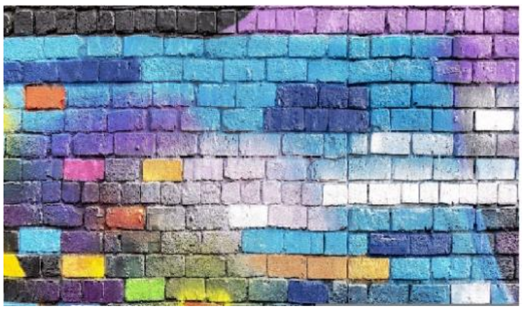


Fig. 16a: Original Image

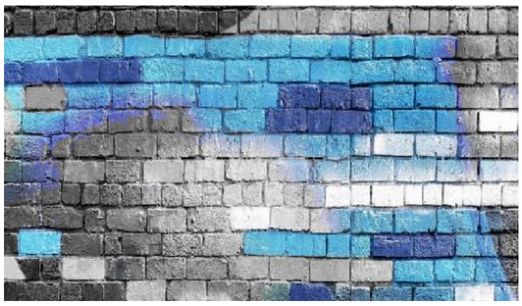


Fig. 16b: Color Filtered Image (blue)

4.4 Change background function:

This function is quite practical and advanced. In our daily life we will use this function quite often due to different restrictions on the background color of the identification photos. The result of this function is clear and is of good help (Fig. 17).

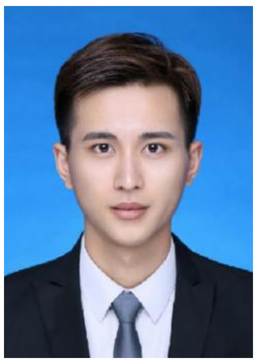


Fig. 17a: Original Image

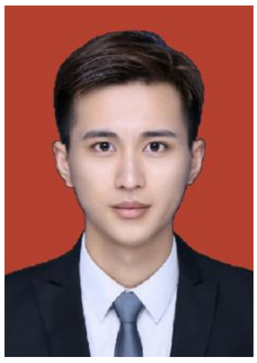


Fig. 17b: Red Background

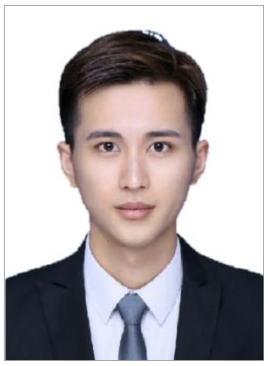


Fig. 17b: White Background

4.5 Various kinds of filters are provided in advanced filters:

There are different kinds of advanced filters that are special and unique. For example, **highlight** and **softlight** are good to deal with portraits. And **beauty** makes the face look better. **Diffuse** has some randomness, which means that users may get different effects by using this filter. And it is also doable to change the parameter for some advanced filters in designing phase. For example, in **beauty** filter, **beautyParam** and **GuassBlurPara** can be customized, which means we can provide many versions of the same filter to meet users' demand. Moreover, **neon** filter looks like that users are drawing using crayon on black grounding. Different filters have different purpose and present different results which are appealing to users (Fig. 18).

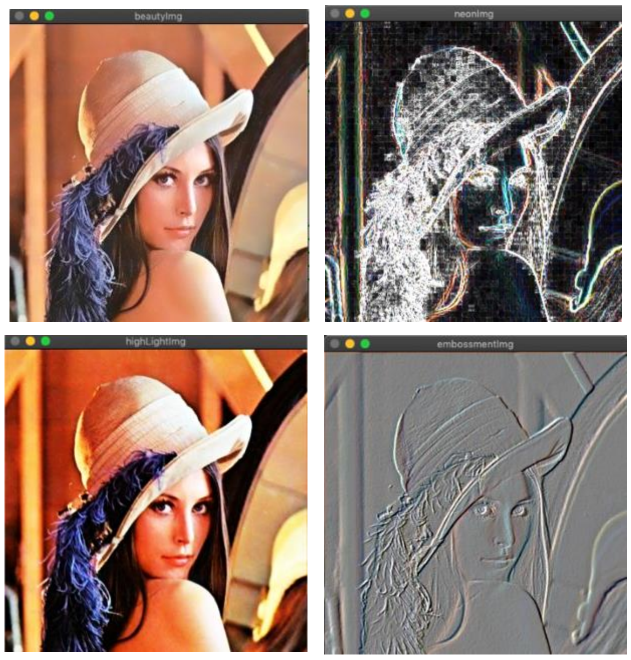


Fig. 18: Some Advanced Filters

5. Conclusion and Future Development

After investigating a great amount of time and energy learning and programming, we finally achieved our goal as stated in the proposal. There are a few lessons we learned:

1. For GUI, we learned how to add the buttons and frames using code instead of using Qt designer, which is very flexible and is good for maintaining the structure.
2. For algorithms, we learned how to use existing classifiers and OpenCV to detect human face and eyes, and combine them with our program.
3. For basic tools, we learned how to obtain main features, such as hue, saturation, value, RGB pixels, via OpenCV. Also, we got to know that the representation of images in OpenCV is matrix.
4. For C++ programming, we used call by reference, deep copy, stack, list, map, superclass etc. These are very important concepts we learned in class. We are happy to apply them in real programming, which helps us to have a better understanding of them.

5. Since our team is composed of six people, it is rather difficult for us to cooperate and divide the work. Still, we had a happy time collaborating with each other and everyone is responsible for their own job. In the process, we learned how important it is to have a clear and scientific division of work in order to avoid conflict.

We think most lessons concerning Qt and OpenCV are special for our project, but the programming part is universal. We would have a better understanding of the knowledge tested in exam such as pointer and superclass if we use them through homework and practice. Therefore, we suggest including more direct interactions with these knowledges through homework.

If more time is allowed, there are a few **future developments** we can make:

1. Use GitHub for version control, which can help us to merge the work easily and timely.
2. The data type transformation between **Mat** and **Qimage** is still a problem that we are confused about. It needs to be further clarified so that the programming logic can be clearer.
3. For mouse control, it would be fancier if we can drag the sticker, the text, or show a crop frame while moving the mouse.
4. For Stickers, a special sticker which can be automatically stick to the image has been developed. This program first detects the face of a human, and add the sticker over the head. However, due to the limitation of time, we did not add this function into our project.
5. For Filters, the transformation center of **expand** and **shrink** are the middle position of the image. However, it would be better to let users define the transformation center. And we can also design additional functions to let users input the parameters in some filters like **beautyParam** in **beauty**.
6. For Text, we can further let the user choose the font and color.
7. For Tools, we can add more basic functions like color thermometry, image curve, vignetting etc. Also, crop by ratio is not included due to limitation of time.

In the end, we would like to thank our friends and USTFs for giving us advice and support. We also want to thank Prof. Huang for pointing out our shortages and affirming our project.