Report submitted by

AI
Ganesh
Venkata Chandra Sekhar Raja
Rahul
Surya Vara Prasad

Table of Contents

Abstract

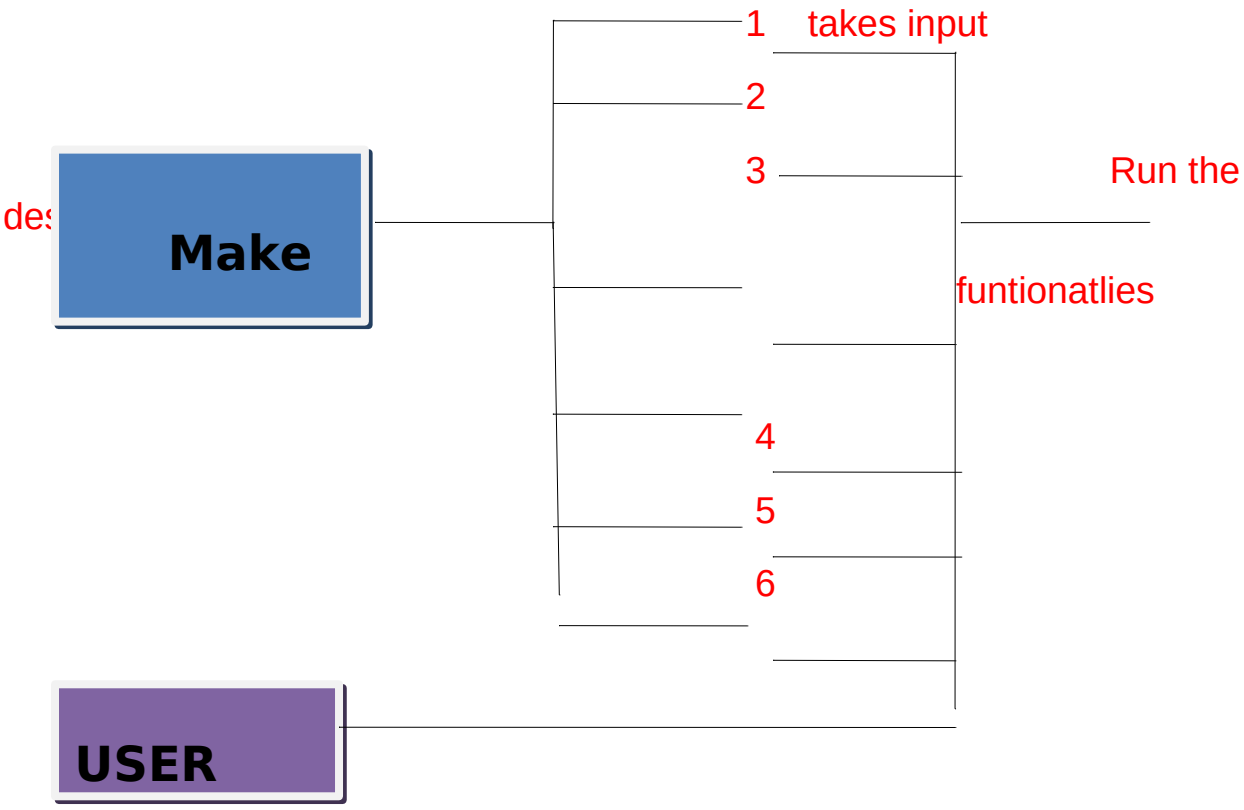
The objective of the project is to develop a small GIMP-like image editor, with basic functions i.e

- **Dilatation / erosion**
- **Resizing**
- **Lighten / darken**
- **Panorama / stitching**
- **Canny edge detection**
- **Face detection**

Introduction:

- In this project we will be building a an application by concatenation of different functionalities in to one main file
- At first we will make file once it generated we have to execute file then once execution takes place.
- User will have to type a number displaying the names on functionalities on the screen once user gives input.
- It will take the input and run desired functionality.
- Once the output is display user have to press any key to go back to the main menu to continue the editing.
- All project was built on CLI(command line interface)

Application:




```
raja@raja-hp: ~/Desktop/project/Project
File  Edit  View  Search  Terminal  Help
raja@raja-hp:~/Desktop/project/Project$ make
rm -f *.o project
g++ -std=c++1y -o project.o -c project.cpp -Wall -O `pkg-config --cflags --libs
opencv`
g++ -o project project.o `pkg-config --cflags --libs opencv`
raja@raja-hp:~/Desktop/project/Project$ ./project
Please choose the action you want to do on the image :
1 : Erosion / Dilation
2 : resizing
3 : lighten / darken
4 : panorama
5 : canny
```

1) Erosion/ Dilasion:

We Apply two very common morphology operators: Dilation and Erosion

The following OpenCV functions:

- **cv::erode**
- **cv::dilate**

A set of operations that process images based on shapes. Morphological operations apply a *structuring element* to an input image and generate an output image.

The most basic morphological operations are two: Erosion and Dilation. They have a wide array of uses, i.e. :

- Removing noise
- Isolation of individual elements and joining disparate elements in an image.
- Finding of intensity bumps or holes in an image

This operations consists of convoluting an image A with some kernel (B), which can have any shape or size, usually a square or circle.

The kernel B has a defined *anchor point*, usually being the center of the kernel.

As the kernel B is scanned over the image, we compute the maximal pixel value overlapped by B and replace the image pixel in the anchor point position with that maximal value. As you can deduce, this maximizing operation causes bright regions within an image to "grow" (therefore the name *dilation*)

```
/** @function Dilation */
void Dilation()
{
  int dilation_elem=0;

  int dilation_size;
  cout<<"Dilation Size : ?"<<endl;
  cin>>dilation_size;

  int dilation_type = 0;
  if( dilation_elem == 0 ){ dilation_type = MORPH_RECT; }
  else if( dilation_elem == 1 ){ dilation_type = MORPH_CROSS; }
  else if( dilation_elem == 2) { dilation_type = MORPH_ELLIPSE; }

  Mat element = getStructuringElement( dilation_type,
                                       Size( 2*dilation_size + 1, 2*dilation_size+1 ),
                                       Point( dilation_size, dilation_size ) );
  /// Apply the dilation operation
  dilate( src, dilation_dst, element );
  imshow( "Dilation", dilation_dst );
}
```

From the above code there are few steps followed

- Load an image (can be BGR or grayscale)
- Create two windows (one for dilation output, the other for erosion)
- Create a set of two Trackbars for each operation:
  - The first trackbar "Element" returns either **erosion_elem** or **dilation_elem**
  - The second trackbar "Kernel size" return **erosion_size** or **dilation_size** for the corresponding operation.
- Every time we move any slider, the user's function **Erosion** or **Dilation** will be called and it will update the output image based on the current trackbar values.

The function that performs the *erosion* operation is **cv::erode** . As we can see, it receives three arguments:

- *src*: The source image
- *erosion_dst*: The output image
- *element*: This is the kernel we will use to perform the operation. If we do not specify, the default is a simple `3x3` matrix. Otherwise, we can specify its shape. For this, we need to use the function **cv::getStructuringElement** :

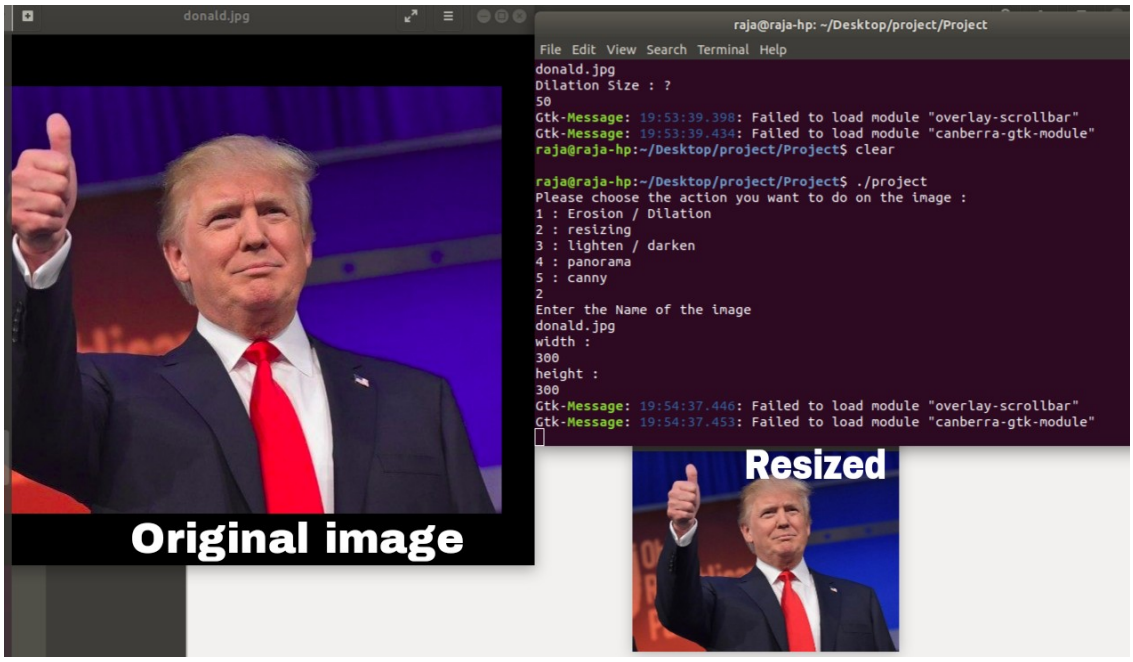
```
169  void Erosion()
170  {
171    int erosion_elem;
172    erosion_elem=1;
173
174    int erosion_size;
175    cout<<"Erosion size : ?"<<endl;
176    cin>>erosion_size;
177
178    int erosion_type = 0;
179    if( erosion_elem == 0 ){ erosion_type = MORPH_RECT; }
180    else if( erosion_elem == 1 ){ erosion_type = MORPH_CROSS; }
181    else if( erosion_elem == 2) { erosion_type = MORPH_ELLIPSE; }
182
183    //![kernel]
184    Mat element = getStructuringElement( erosion_type,
185                                         Size( 2*erosion_size + 1, 2*erosion_size+1 ),
186                                         Point( erosion_size, erosion_size ) );
187    //![kernel]
188
189    /// Apply the erosion operation
190    erode( src, erosion_dst, element );
191    imshow( "Erosion", erosion_dst );
192  }
193
```

2)Resize:

The function resize resizes the image src down to or up to the specified size. Note that the initial dst type or size are not taken into account. Instead, the size and type are derived from the src,``dsize``,``fx`` , and fy . If you want to resize src so that it fits the pre-created dst , you may call the function as follows:

```
Mat dst;//dst image

Mat src;//src image

resize(src,dst,size

resize(src, img_dst, Size(), w, h);//resize image with width and height will be give
by user

    imshow("Resize",img_dst);
```

output:



3)lighten/darken:

The two commonly used point processes are *multiplication* and *addition* with a constant:

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

The parameters $\alpha>0$ and β are often called the *gain* and *bias* parameters; sometimes these parameters are said to control *contrast* and *brightness*respectively.

You can think of $f(x)$ as the source image pixels and $g(x)$ as the output image pixels. Then, more conveniently we can write the expression as:

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

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

Code Explanation

1. We begin by creating parameters to save α and β to be entered by the user:

```
double alpha = 1.0; /*< Simple contrast control */

int beta = 0; /*< Simple brightness control */
```

2. We load an image using **cv::imread** and save it in a Mat object:

```
String imageName("../data/lena.jpg"); // by default

if (argc > 1)

{

imageName = argv[1];

}

Mat image = imread( imageName );
```

3. Now, since we will make some transformations to this image, we need a new Mat object to store it. Also, we want this to have the following features:
   - Initial pixel values equal to zero
   - Same size and type as the original image

```
Mat new_image = Mat::zeros( image.size(), image.type() );
```

We observe that **cv::Mat::zeros** returns a Matlab-style zero initializer based on *image.size()* and *image.type()*

4. Now, to perform the operation $g(i,j)=\alpha \cdot f(i,j)+\beta$ we will access to each pixel in image. Since we are operating with BGR images, we will have three values per pixel (B, G and R), so we will also access them separately. Here is the piece of code:

```
for( int y = 0; y < image.rows; y++ ) {
for( int x = 0; x < image.cols; x++ ) {
for( int c = 0; c < 3; c++ ) {
new_image.at<Vec3b>(y,x)[c] =
saturate_cast<uchar>( alpha*( image.at<Vec3b>(y,x)[c] ) + beta );
}
}
}
```

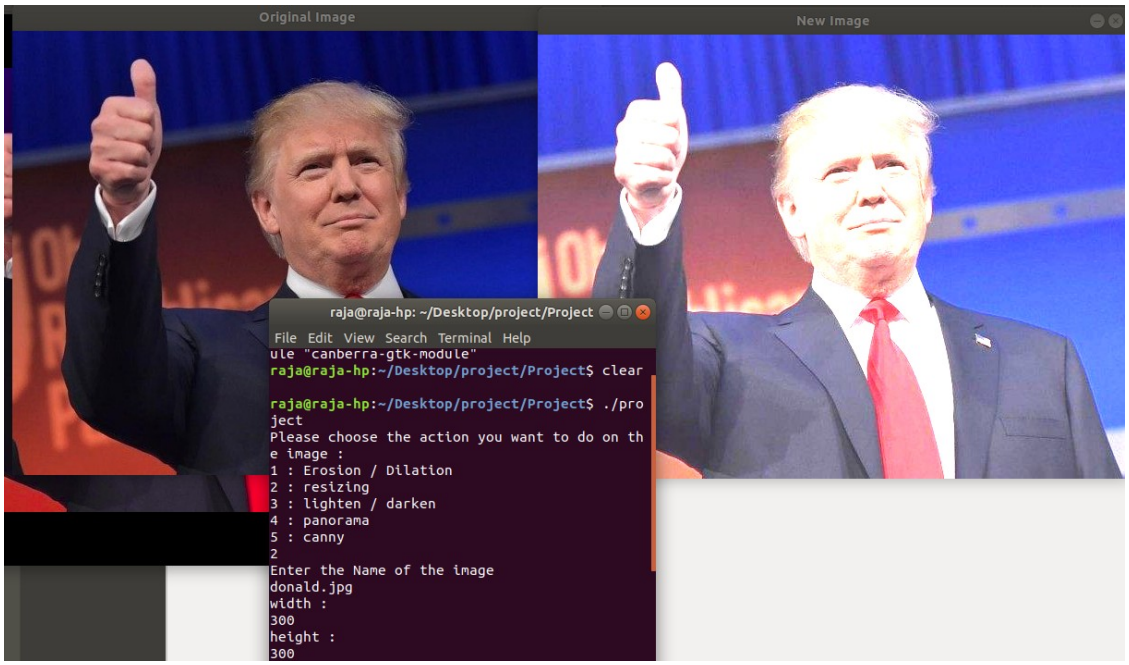
Notice the following:

- To access each pixel in the images we are using this syntax: *image.at<Vec3b>(y,x)[c]* where *y* is the row, *x* is the column and *c* is R, G or B (0, 1 or 2).
- Since the operation $\alpha \cdot p(i,j)+\beta$ can give values out of range or not integers (if α is float), we use **cv::saturate_cast** to make sure the values are valid.

5. Finally, we create windows and show the images, the usual way.

```
namedWindow("Original Image", WINDOW_AUTOSIZE);
namedWindow("New Image", WINDOW_AUTOSIZE);
imshow("Original Image", image);
imshow("New Image", new_image);
waitKey();
```

output:


Original Image
New Image
raja@raja-hp: ~/Desktop/project/Project
File Edit View Search Terminal Help
ule "canberra-gtk-module"
raja@raja-hp:~/Desktop/project/Project$ clear
raja@raja-hp:~/Desktop/project/Project$ ./pro
ject
Please choose the action you want to do on th
e image :
1 : Erosion / Dilation
2 : resizing
3 : lighten / darken
4 : panorama
5 : canny
2
Enter the Name of the image
donald.jpg
width :
300
height :
300

5)Panarama:

set the high-level stitching API for stitching provided by

a. **cv::Stitcher**

learn how to use preconfigured Stitcher configurations to stitch images

```
Mat pano;
Ptr<Stitcher> stitcher = Stitcher::create(mode, try_use_gpu);
Stitcher::Status status = stitcher->stitch(imgs, pano);
if (status != Stitcher::OK)
{
cout << "Can't stitch images, error code = " << int(status) << endl;
return -1;
}
```

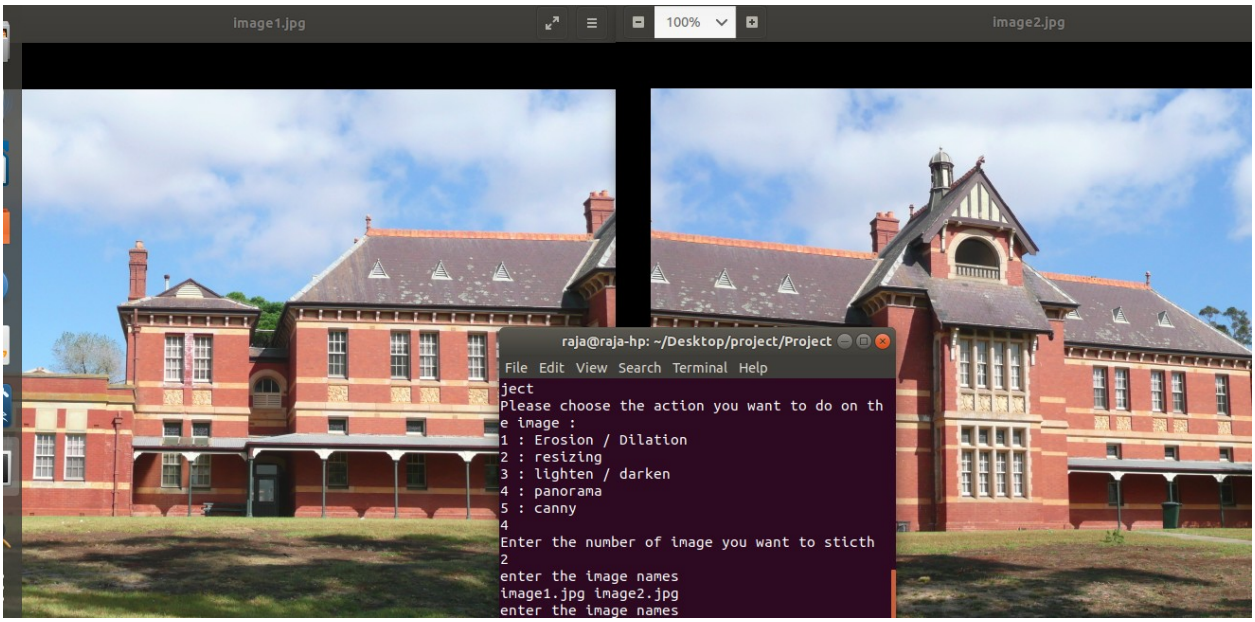
A new instance of stitcher is created and the **cv::Stitcher::stitch** will do all the hard work.

cv::Stitcher::create can create stitcher in one of the predefined configurations (argument `mode`). See **cv::Stitcher::Mode** for details. These configurations will setup multiple stitcher properties to operate in one of predefined scenarios. After you create stitcher in one of predefined configurations you can adjust stitching by setting any of the stitcher properties.

If you have cuda device **cv::Stitcher** can be configured to offload certain operations to GPU. If you prefer this configuration set `try_use_gpu` to true. OpenCL acceleration will be used transparently based on global OpenCV settings regardless of this flag.

Stitching might fail for several reasons, you should always check if everything went good and resulting pano is stored in `pano`. See **cv::Stitcher::Status**documentation for possible error codes.

Output:



Result:

5)Canny Edge Detection

he *Canny Edge detector* was developed by John F. Canny in 1986. Also known to many as the *optimal detector*, Canny algorithm aims to satisfy three main criteria:

- **Low error rate:** Meaning a good detection of only existent edges.
- **Good localization:** The distance between edge pixels detected and real edge pixels have to be minimized.
- **Minimal response:** Only one detector response per edge.

How it Works

At first asks the user to enter a numerical value to set the lower threshold for our *Canny Edge Detector* (by means of a Trackbar)

- Applies the *Canny Detector* and generates a **mask** (bright lines representing the edges on a black background).
- Applies the mask obtained on the original image and display it in a window.

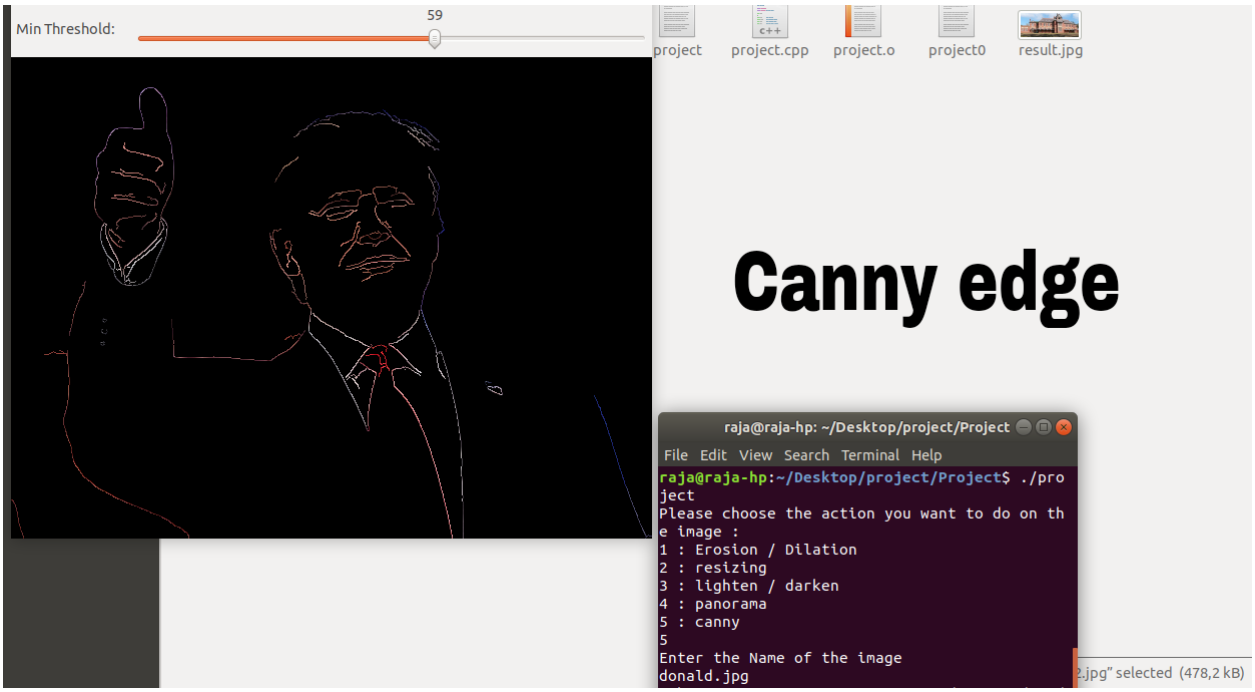
Main function : void canny()

{

```
 /// Create a matrix of the same type and size as src (for dst)
dst.create( src.size(), src.type() );
//![create_mat]

//![convert_to_gray]
cvtColor( src, src_gray, COLOR_BGR2GRAY );
//![convert_to_gray]

//![create_window]
```

```
  namedWindow( window_name, WINDOW_AUTOSIZE );

  //![create_window]


  //![create_trackbar]

  /// Create a Trackbar for user to enter threshold

  createTrackbar( "Min Threshold:", window_name, &lowThreshold, max_lowThreshold,
CannyThreshold );

  //![create_trackbar]


  /// Show the image

  CannyThreshold(0, 0);


 }
```

out put:



6. Face detection(Advanced)

One of the mainstream methods of face detection is class haar+adaboosting, which is also used in OpenCV. This method can be extended to the detection of rigid objects, the premise is to train the cascade classifier (for example, with class haar features), once the training data is completed, directly call opencv class CascadeClassifier, use it a few simple of member functions can complete the detection function.

Main function:

```
int
main(
)
        {


            CascadeClassifier faceDetector;
            std::string faceCascadeFilename = "haarcascade_frontalface_default.xml";



            try{
                faceDetector.load(faceCascadeFilename);
            }
            catch (cv::Exception e){}
            if (faceDetector.empty())
            {
                std::cerr << "Face detection can't be loaded (";
                std::cerr << faceCascadeFilename << ")!" << std::endl;
                exit(1);
            }



            VideoCapture camera(0);
            while (true)
            {
                Mat camerFrame;
                camera >> camerFrame;
                if (camerFrame.empty())
                {
                    std::cerr << "Can't catch image in camera" << std::endl;
                    getchar();
```

```
            exit(1);
        }
        Mat displayedFrame(camerFrame.size(),CV_8UC3);



        Mat gray;
        Pic2Gray(camerFrame, gray);




        Mat equalizedImg;
        equalizeHist(gray, equalizedImg);

        //人脸检测用 Cascade Classifier::detectMultiScale 来进行人脸检测

        int flags = CASCADE_FIND_BIGGEST_OBJECT|CASCADE_DO_ROUGH_SEARCH;    //
只检测脸最大的人
        //int flags = CASCADE_SCALE_IMAGE;    //检测多个人
        Size minFeatureSize(30, 30);
        float searchScaleFactor = 1.1f;
        int minNeighbors = 4;
        std::vector<Rect> faces;
        faceDetector.detectMultiScale(equalizedImg, faces, searchScaleFactor,
minNeighbors, flags, minFeatureSize);



        cv::Mat face;
        cv::Point text_lb;
        for (size_t i = 0; i < faces.size(); i++)
        {
            if (faces[i].height > 0 && faces[i].width > 0)
            {
                face = gray(faces[i]);
                text_lb = cv::Point(faces[i].x, faces[i].y);
                cv::rectangle(equalizedImg, faces[i], cv::Scalar(255, 0, 0),
1, 8, 0);
                cv::rectangle(gray, faces[i], cv::Scalar(255, 0, 0), 1, 8, 0);
                cv::rectangle(camerFrame, faces[i], cv::Scalar(255, 0, 0), 1,
8, 0);
```

```
                    }
                }


                imshow("Histogram homogenization", equalizedImg);
                imshow("Grayscale", gray);
                imshow("Original", camerFrame);

                waitKey(20);
            }

            getchar();
            return 0;
        }
```

Conclusion:

In the current project we build an image editor with different general functionalities and advanced functionalities on opencv.we learned all the functionalities by this project. As we found out that it is less likely to be adopted as a gui interface a common user who is not familiar with CLI is less likely to use this application widely but if the user is familiar with CLI it will be easy to use.

References:

https://docs.opencv.org