Creating a Graphical User Interface(GUI) for processing images

Howard Edwards, Michael Micros, Jonathon Rigney, Megan Rowland

***Abstract(Howard Edwards)* —**
This report covers the development of a graphical user interface, GUI, used for image processing. The graphical user interface allows for the reading and writing of 8-bit grayscale images for processing. Image processing done by the GUI is image overlaying, histogram equalization, and the image adjustment of brightness and contrast. After the completion of image processing, the interface displays the processed images. All image processing is initiated completely via the interfaces widgets and results displayed simultaneously.

***Index Terms* — Gray-scale, Image analysis, Image processing**

I. Introduction(Jonathon Rigney)

The goal of the research being done is to design, build, test and demonstrate an integrated system that performs real-time video processing based on field-programmable gating arrays (FPGAs) and is controlled via a graphical user interface (GUI) on a PC. The current module focuses on the creation of a graphical user interface using Qt software. This interface allows the user to select a grayscale bitmap image and adjust the contrast and brightness with sliders. The user can also select an image to overlay with the original image, adding things like company logos and subtitles.

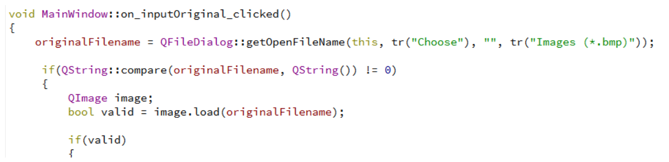
II. Design ()

A. Summary of Design–

The overall design consists of a Qt program that provides a GUI with which the user will interact. The GUI contains several buttons, each of which performs a specific action. These actions include loading in and saving images, performing histogram equalization and overlaying the uploaded image with another image of the users choice. Finally, the design also incorporates sliders that allow for the adjustment of image contrast and brightness.

B. Detail Description

Loading an Image: In order to load an image to the GUI, the Choose and Image button must be clicked. By programming the following action into the button the user is prompted to select an image.



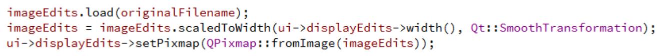
```
void MainWindow::on_inputOriginal_clicked()
{
    originalFilename = QFileDialog::getOpenFileName(this, tr("Choose"), "", tr("Images (*.bmp)"));

     if(QString::compare(originalFilename, QString()) != 0)
     {
         QImage image;
         bool valid = image.load(originalFilename);

         if(valid)
         {
```

Figure 1: Code that promts user to select an Image and loads it to a QImage object

The QImage object is then displayed in the Label object which allows it to be visible to the user.



```
imageEdits.load(originalFilename);
imageEdits = imageEdits.scaledToWidth(ui->displayEdits->width(), Qt::SmoothTransformation);
ui->displayEdits->setPixmap(QPixmap::fromImage(imageEdits));
```

Figure 2: Code that displays image on GUI using Label object

A similar process is used whenever it is necessary to display an image in the GUI. Upon loading in an image, the histogram equalization of that image is performed automatically according to the method used in the paper by Edwards et al [1] (Milestone 1 Report). Also the maximum and minimum pixel values of the uploaded image are calculated. These values will be used in adjusting the brightness and contrast of the image.

Overlaying: In order to perform image overlaying, an action is assigned to the Choose and Overlay Image button on the GUI. When pressed, the user will be prompted to select an image to be overlaid on the original image that the user has already uploaded. Upon selecting the image, the image overlay process takes place according to Edwards et al [1].The result of that function is out1.bmp, which is automatically loaded and displayed in the proper label.

Contrast adjustment using Slider widget: The slider widget has a signal named value changed which is activated whenever the slider is moved. The slider represents a value from 0 to 99 with 0 being all the way left and 99 being all the way to the right. To calculate the changes in contrast to each pixel in the image as the slider is moved to the left or right the following formula is used:

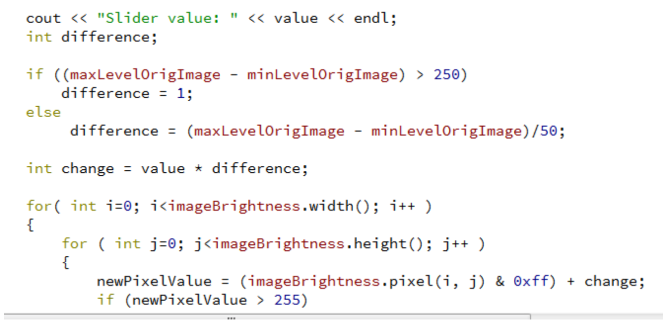
$$\text{newPV} = \frac{\text{newMin+ (newMax-newMin)} \times \text{(origPix - oldMin)}}{\text{(oldMax-oldMin)}}$$

where imageEdits.pixel(i, j) = original image pixel value, newPV = contrast changed pixel, newMin = minimum pixel value in the new range, newMax = maximum pixel value in the new range, oldMin = minimum pixel level in the original image, oldMax = maximum pixel level in the original image.

The newMinLevel and the newMaxLevel proportionally increase as the slider is moved to the right and decreases as the slider is moved to the left. This allows the contrast of the image to be enhanced by utilizing a greater range of pixel values and vice versa lessen by utilizing a smaller range of pixel values. If the range of pixel values are greater than 50 to 205, the newMinLevel is set to 50 and the newMaxLevel is set to 205. This allows an increase in range of pixel by one for each increment of the slider.

Brightness adjustment using Slider widget: Similarly to the contrast adjustment, the value of the slider is used to calculate the new value of each pixel in the image to be displayed. The overall objective is to add a certain value to each image pixel according to the position of the slider (and consequently the value that it represents). As can be seen in the Figure below, the constant value that is added to all pixels is described by the equation:

$$\text{change} = \frac{value \times (maxLevelOrigImage - minLevelOrigImage)}{50}$$



```
cout << "Slider value: " << value << endl;
int difference;

if ((maxLevelOrigImage - minLevelOrigImage) > 250)
    difference = 1;
else
     difference = (maxLevelOrigImage - minLevelOrigImage)/50;

int change = value * difference;

for( int i=0; i<imageBrightness.width(); i++ )
{
    for ( int j=0; j<imageBrightness.height(); j++ )
    {
        newPixelValue = (imageBrightness.pixel(i, j) & 0xff) + change;
        if (newPixelValue > 255)
```

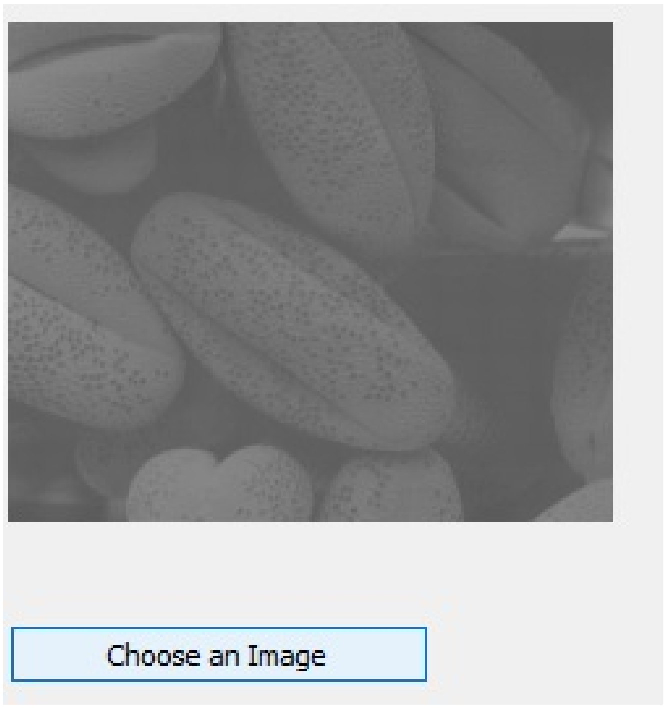
Figure 3: Code demonstrating the process of brightness adjustment

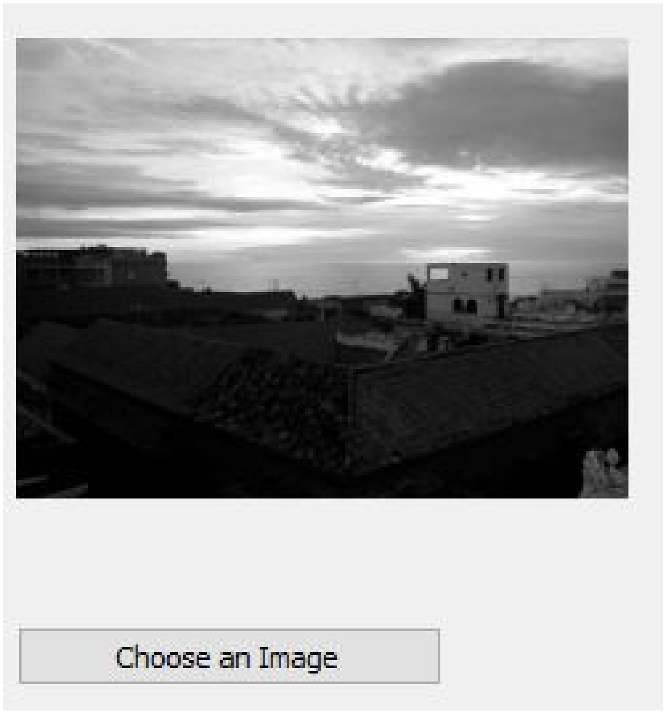
Saving an image: In similar fashion to uploading an image, saving an image is performed by clicking the button that corresponds to the image (Label object) to be saved to file.This is accomplished using the Qimage::save() function that requires the filename, format and quality factor to be specified. All images are saved in appropriately named files, specifically : "HistogramEqualizedImage.bmp", "OverlaidImage.bmp", "ContrastEditsImage.bmp", "BrightnessEditsImage.bmp".

III. Evaluation ()

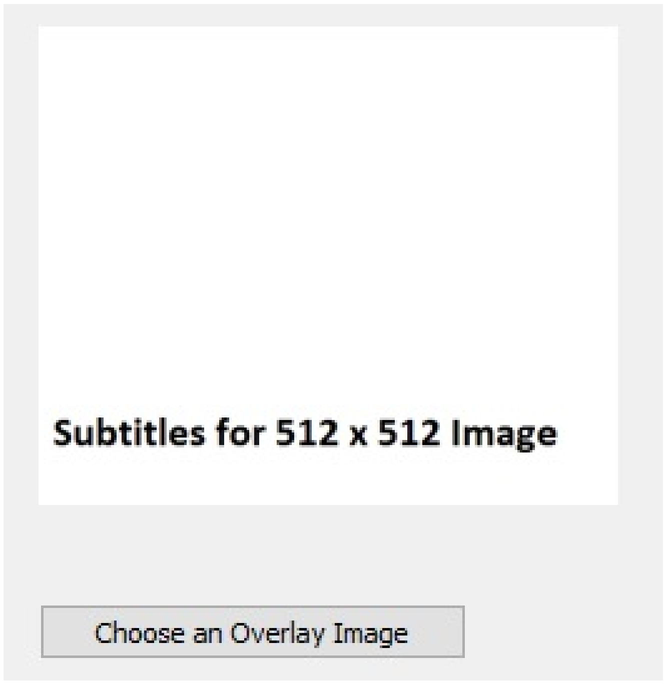
he program functions and produces the outcomes as expected.

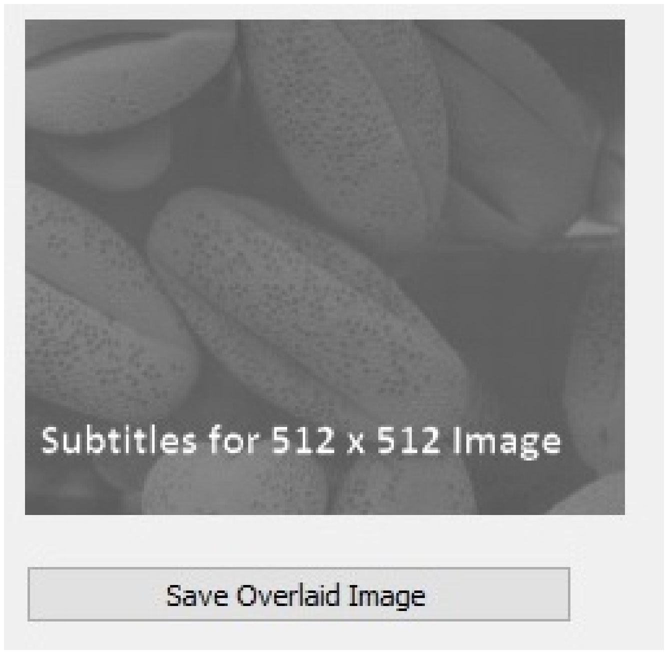
First, the GUI allows the user to load an 8-bit bitmap original image. Once the user clicks on the Choose an Image button and selects the image, the original image loads in the view above the button:

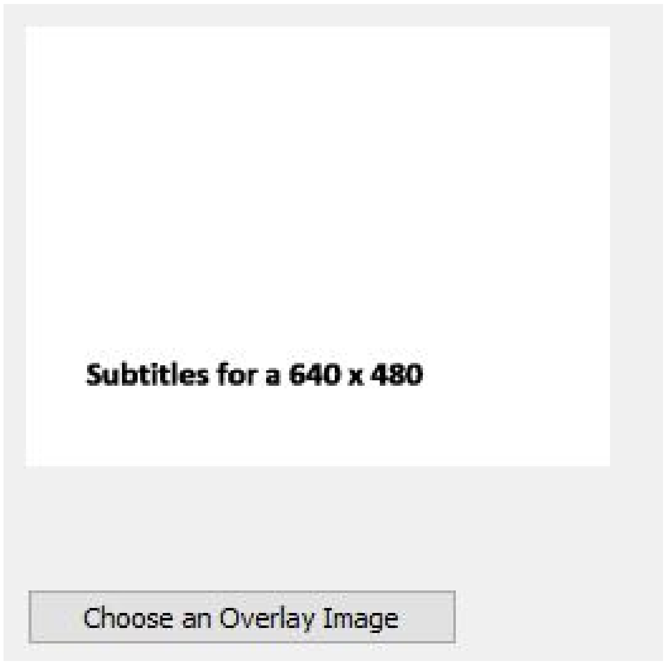


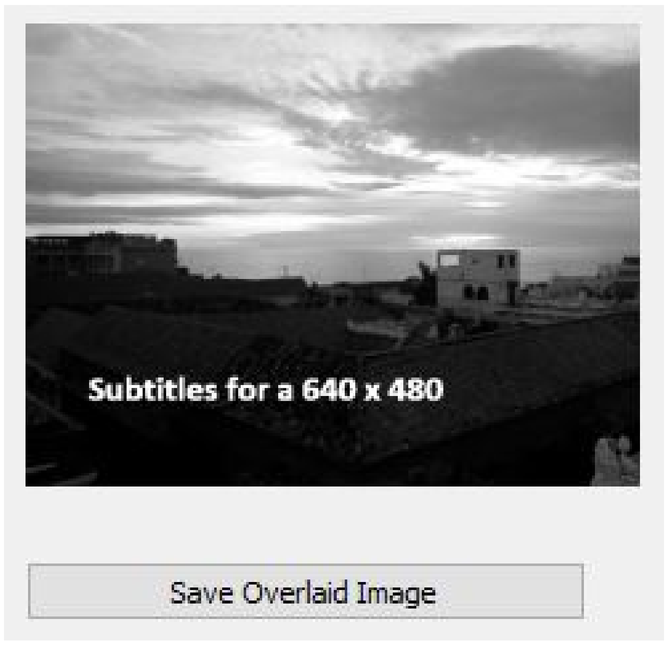


Second, the GUI allows the user to load an 8-bit bitmap overlay image. Once the user clicks on the Choose an Overlay Image button and select the image, the overlay image loads in the view above the button:



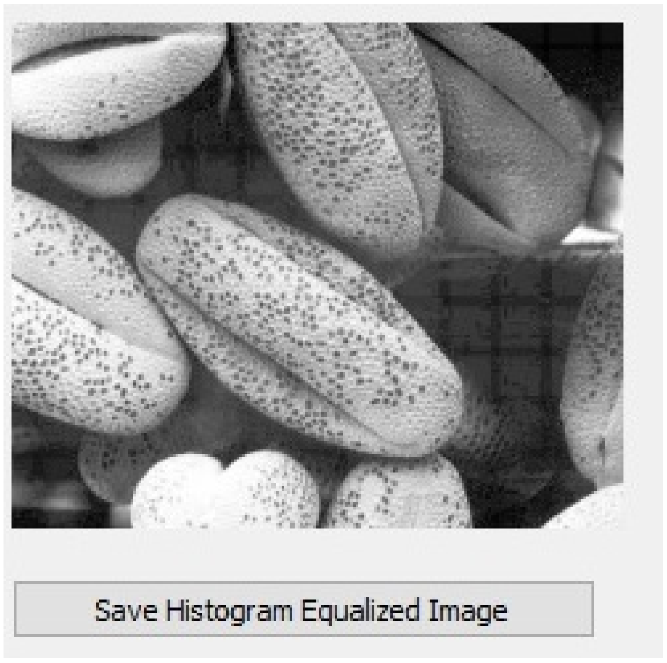


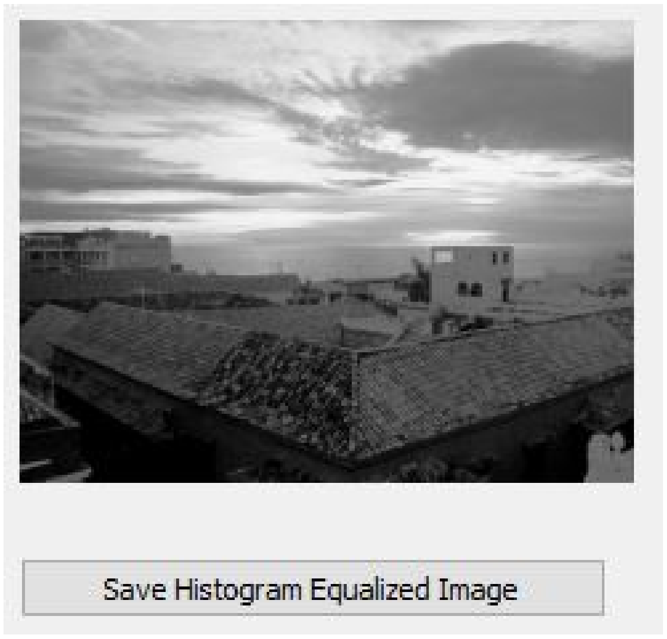




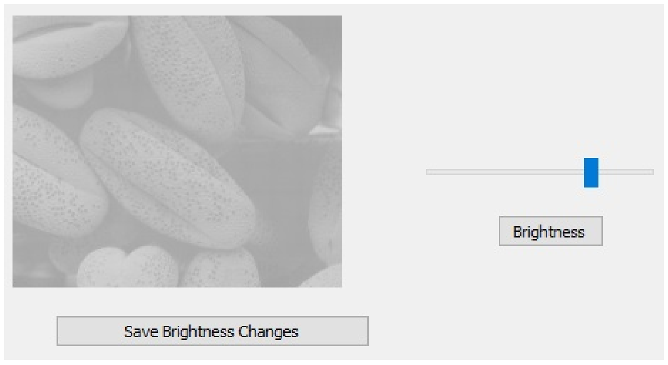
Third, the GUI generates and displays, below the overlay image load function, an image by overlaying the overlay image over the original image. Below the image is a button to Save Overlaid Image to a file. Clicking on this button saves the overlaid image:

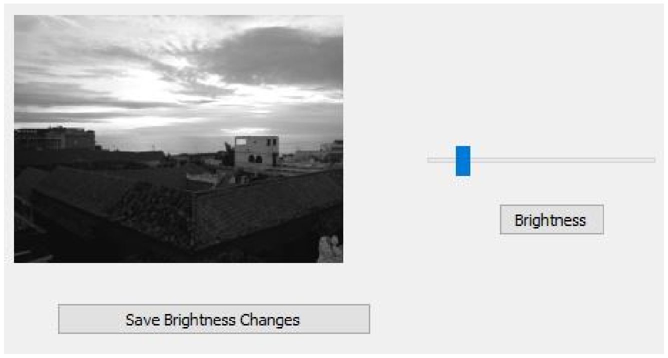
Fourth, the GUI generates and displays an image from the histogram equalization of the original image. Below the image is a button to Save Histogram Equalized Image to a file. Clicking on this button saves the histogram equalized image:



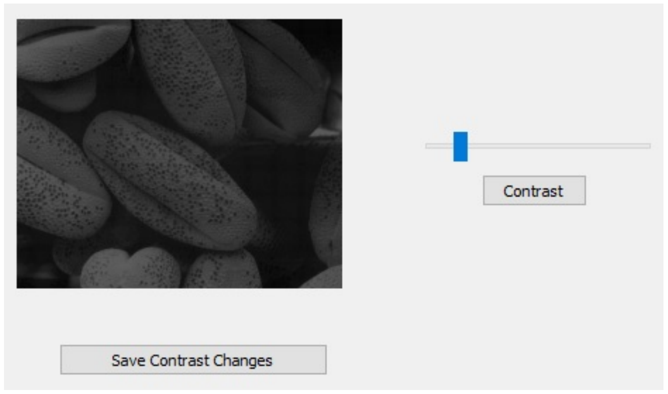


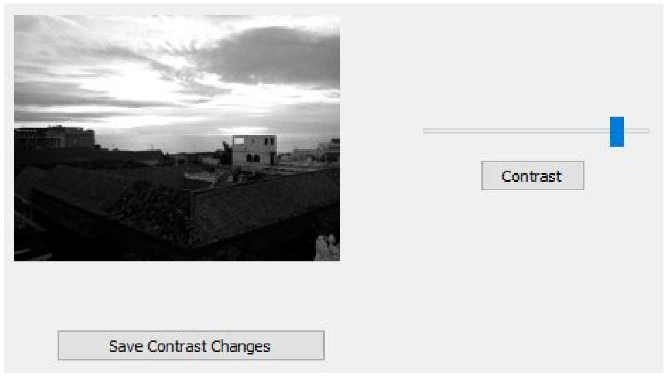
Fifth, the GUI offers a slider to adjust the brightness of the original image. Move the slider to manually adjust the contrast of the original image. Below the image is a button to Save Brightness Changes to a file. Click on this button saves the changes in a brightness file. As can be seen below, when the slider is moved to the left the brightness decreases and when the slider is moved to the right the brightness increases.





Sixth, the GUI offers a slider to adjust the contrast of the original image. Move the slider to manually adjust the contrast of the original image. Below the image is a button to Save Contrast Changes to a file. Clicking on this button saves the changes in contrast to a file. As can be seen below, the contrast of the image decreases when the slider is moved to the left and the contrast increases as the slider is moved to the right.





IV. Discussion ()

Based on the results in the Evaluation section, the GUI functions as expected. There is not much room for improvement for the GUI. The layout is self explanatory, all of the different functions respond very quickly, and the images are large enough to view efficiently without taking up too much landscape. The contrast and brightness sliders seem to work best with an image that occupies a limited range of values (for example 160 to 230) rather than the entire 255 length spectrum. One way in which to improve the GUI is to make the layout a bit easier on the eye, but that will be left up to the Human Factors team to optimize. Also more effective algorithms to perform the contrast adjustment could be implemented, but the speed of the application may be negatively impacted.

V. Appendix