HACETTEPE UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING

**BBM 413 – BBM415**

# COMPUTER IMAGE TERM PROJECT REPORT

**Mustafa Kollu**
**21627485**

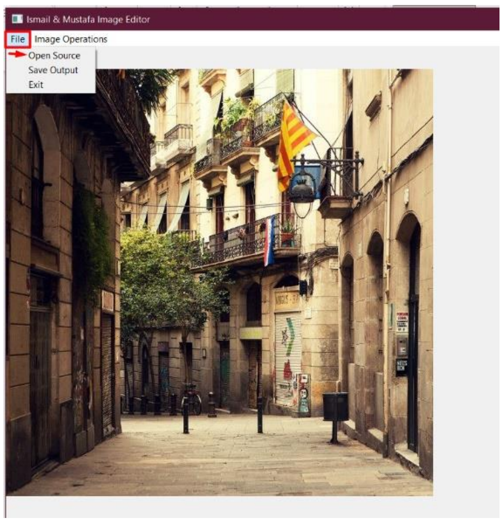
**İsmail Ateş**
**21626953**

## 1. Introduction

In this project, we will put various image applications that we learned in the BBM413 Computer Image course into a real-time application. Within the scope of this project, we will perform the following functions: upload and display image, save image, blur/unblur image, grayscale image, crop image, flip image, mirror image, rotate image, invert image color, change color balance, adjust image brightness, adjust image contrast, adjust image saturation, add noise to the image, detect the edges of the image. In this project, we will make an image application with a graphical user interface (GUI) with the Python language and Python framework QT (desktop application making tool). Thus, a user can easily perform the operations with the functions described above. In our application, all our functions work without any problems. We also benefited from different libraries of Python such as OpenCV for the project.

## 2. Functions

### 2.1 Load Image and Show Image

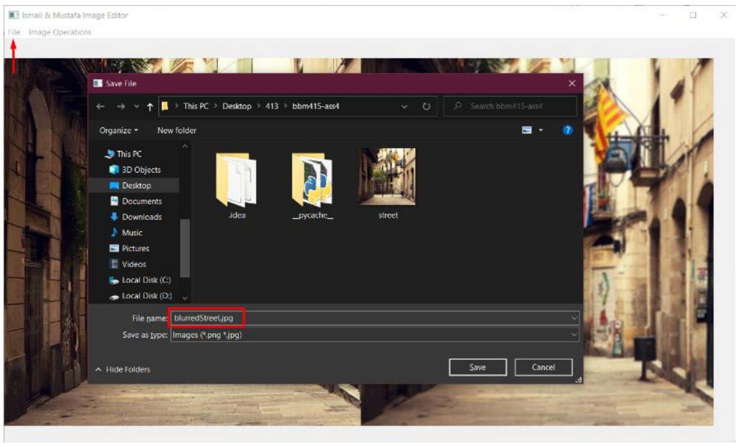


```python
def openSource(imageLabel, stateManager, outputImageLabel):
    dialogMenu = QtWidgets.QDialog()
    sourcePath, _ = QtWidgets.QFileDialog.getOpenFileName(dialogMenu,
                                                          "Open Image", "", "Image (*.jpg *.png)")
    if sourcePath != '':
        imageLabel.setPixmap(QtGui.QPixmap(sourcePath))
        stateManager.importImage(sourcePath)
        stateManager.outputSource = ''
        outputImageLabel.setPixmap(QtGui.QPixmap())
```

Photo-1

As we can see in Photo-1, we can show our application by selecting the jpg and png files we want from the pop-up window. We used QTGui and QPixmap, one of the ready-made functions of QT, as the code.

### 2.2 Save Image



File name: blurredStreet.jpg
Save as type: Images (*.png *.jpg)

```python
def saveImageAs(stateManager):
    if checkOutputPhotoExist(stateManager):

        dialogMenu = QtWidgets.QDialog()
        Image = stateManager.outputSource
        fileName = QtWidgets.QFileDialog.getSaveFileName(dialogMenu, "Save File",
                                                         "",
                                                         "Images (*.png *.jpg)")
        Image.save(fileName[0])
    else:
        warnMessage("Output Image can not be null!!")
```

As we can see in Photo-2, we can save the desired output in jpg or png format to the file directory we want without any problems. We used Qt's QtWidget ready functions as code.

Photo-2

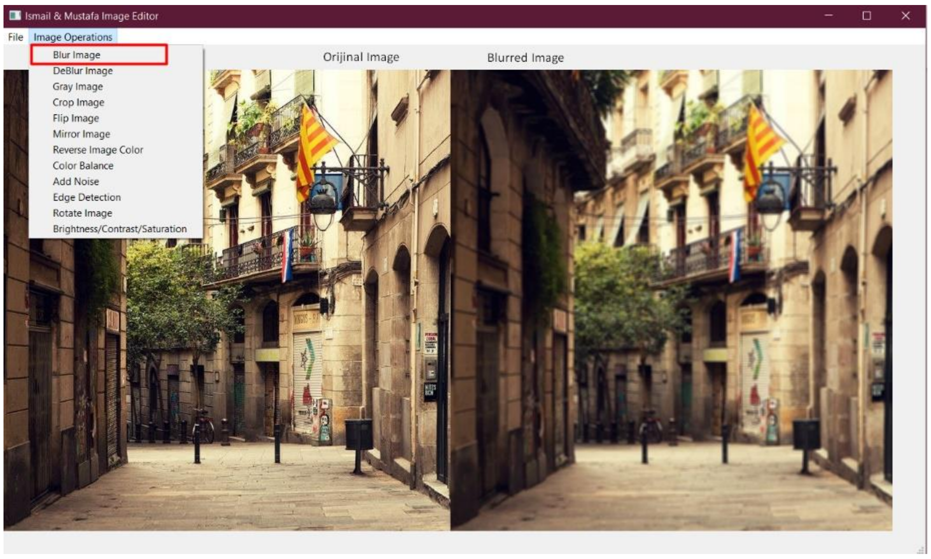


Photo-3

## 2.3 Blur Image

As we can see in Photo-3, we are blurring our original image. We used the OpenCV library as the filter. In this method, instead of a box filter, a Gaussian kernel is used. We should specify the width and height of the kernel which should be positive and odd. Gaussian blurring is highly effective in removing Gaussian noise from image.

```
def blurImage(outputImageLabel, stateManager):
    if checkInputPhotoExist(stateManager):
        if stateManager.outputSource != '':
            inputImage = QImageToCvMat(stateManager.outputSource)
        else:
            inputImage = imread(stateManager.inputSource)

        outputData = opencv.blur(inputImage, (5, 5))
        outputQImage = qimage2ndarray.array2qimage(outputData, normalize=True)
        outputImageLabel.setPixmap(QtGui.QPixmap(outputQImage))
        stateManager.imageOperation(
            outputQImage)
    else:
        warnMessage("Please open a image!")
```

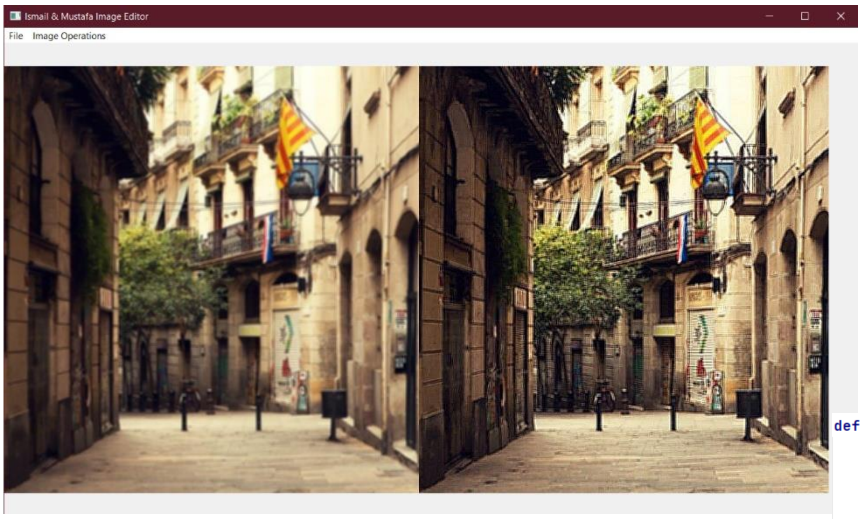


Photo-4

## 2.4 Deblur Image

As we saw in Photo-4, we are trying to convert our blurred image to its original state (Deblur). As a filter, we applied the

[[-1, -1, -1], [-1,9, -1], [-1, -1, -1]] sharpening filter.

Our kernel is 3X3 matrix which we define as per our need to slide over the image for convolution operation. We use cv2.filter2D for OpenCV provides us with a function called filter2D to convolve a kernel with an image.

```
def deBlur(outputImageLabel, stateManager):
    if checkInputPhotoExist(stateManager):
        if stateManager.outputSource != '':
            inputImage = QImageToCvMat(stateManager.outputSource)
        else:
            inputImage = imread(stateManager.inputSource)

        kernel = np.array([[-1,-1,-1], [-1,9,-1], [-1,-1,-1]])
        outputData = opencv.filter2D(src=inputImage, ddepth=-1, kernel=kernel)
        outputQImage = qimage2ndarray.array2qimage(outputData, normalize=True)
        outputImageLabel.setPixmap(QtGui.QPixmap(outputQImage))
        stateManager.imageOperation(
            outputQImage)
    else:
        warnMessage("Please open a image!")
```

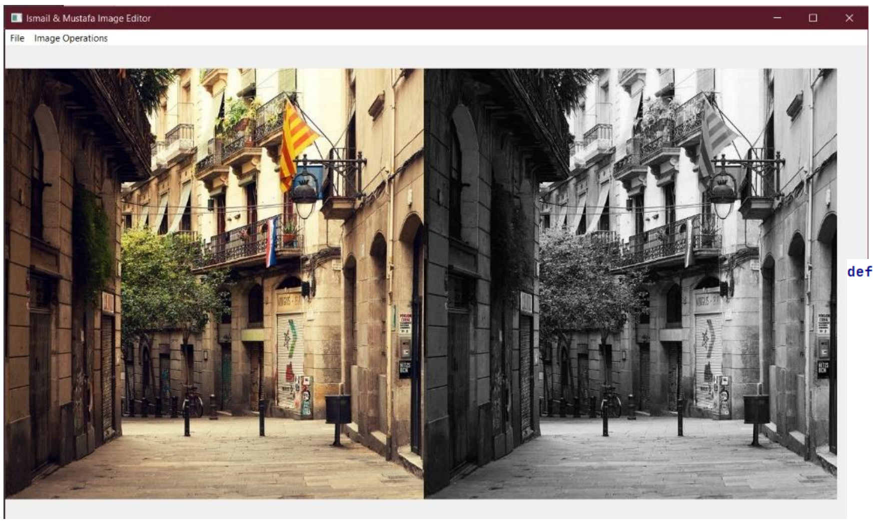


Photo-5

## 2.5 Grayscale Image

As we see in Photo-5, we turn our original image to grey level. As a filter, we applied OpenCV's COLOR_BGR2GRAY filter.

```
def rgbToGrayscale(outputImageLabel, stateManager):
    if checkInputPhotoExist(stateManager):
        if stateManager.outputSource != '':
            inputImage = QImageToCvMat(stateManager.outputSource)
        else:
            inputImage = imread(stateManager.inputSource)

        outputData = opencv.cvtColor(inputImage, opencv.COLOR_BGR2GRAY)
        outputQImage = qimage2ndarray.gray2qimage(outputData, normalize=True)
        outputImageLabel.setPixmap(QtGui.QPixmap(outputQImage))
        stateManager.imageOperation(
            outputQImage)
    else:
        warnMessage("Please open a image!")
```

CV_BGR2GRAY: Lower right corner: 0.1140 * 163 + 0.5870 * 182 + 0.2989 * 203 ≈ 186
CV_RGB2GRAY: Lower right corner: 0.1140 * 203 + 0.5870 * 182 + 0.2989 * 163 ≈ 179
We try both of them and we saw “CV_BGR2GRAY” is giving better result. So we use “CV_BGR2GRAY”. Because as you can see above there are different RGB equivalents.

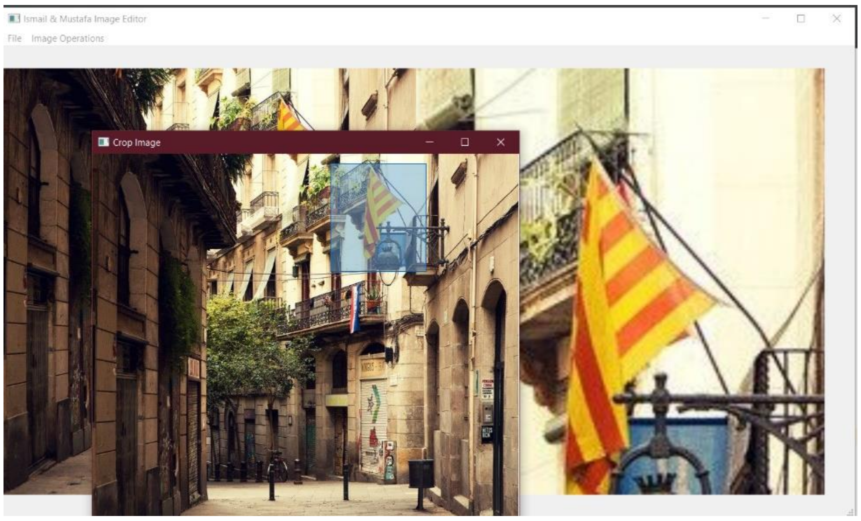


Photo-6

## 2.6 Crop Image

We cut the part of our original picture that we saw in Photo-6 and transform it into a different photograph. We used QT's Qpixmap method as a code. In this method, we keep the points of the geometric area clipped by the mouse, cut the relevant area from the photo, and transform it into a new picture.

```
def mousePressEvent(self, eventQMouseEvent):
    self.originQPoint = eventQMouseEvent.pos()
    self.currentQRubberBand = QRubberBand(QRubberBand.Rectangle, self)
    self.currentQRubberBand.setGeometry(QRect(self.originQPoint, QtCore.QSize()))
    self.currentQRubberBand.show()

def mouseMoveEvent(self, eventQMouseEvent):
    self.currentQRubberBand.setGeometry(QRect(self.originQPoint, eventQMouseEvent.pos()).normalized())

def mouseReleaseEvent(self, eventQMouseEvent):
    self.currentQRubberBand.hide()
    currentQRect = self.currentQRubberBand.geometry()
    self.currentQRubberBand.deleteLater()
    cropQPixmap = self.pixmap().copy(currentQRect)
    self.outputImageLabel.setPixmap(QtGui.QPixmap(cropQPixmap))
    self.stateManager.imageOperation(
        cropQPixmap.toImage())
```

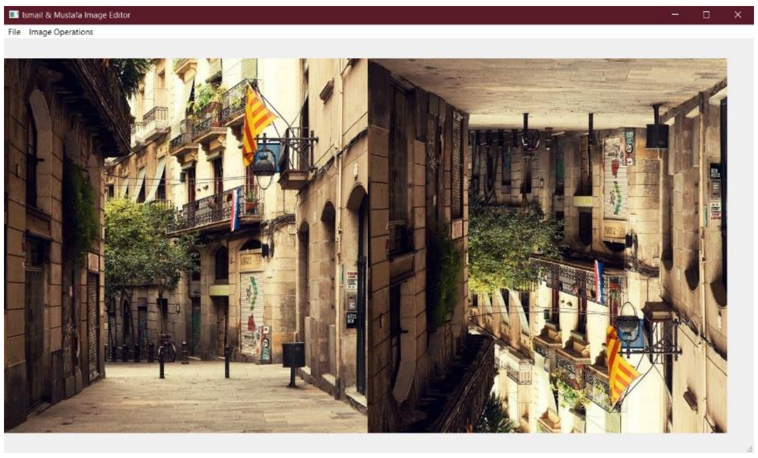


Photo-7

## 2.7 Flip Image

We get an image as if we put a mirror under our original picture that we saw in Photo-7. We used the flip method of the OpenCV library as a filter. This method is used to flip a 2D array. The cv::flip function rotates a 2D array around vertical, horizontal, or both axes based on the input value we provide.

```
def flipImage(outputImageLabel, stateManager):
    if checkInputPhotoExist(stateManager):
        if stateManager.outputSource != '':
            inputImage = QImageToCvMat(stateManager.outputSource)
        else:
            inputImage = imread(stateManager.inputSource)

        outputData = opencv.flip(inputImage, 0)
        outputQImage = qimage2ndarray.array2qimage(outputData, normalize=True)
        outputImageLabel.setPixmap(QtGui.QPixmap(outputQImage))
        stateManager.imageOperation(
            outputQImage)
    else:
        warnMessage("Please open a image!")
```

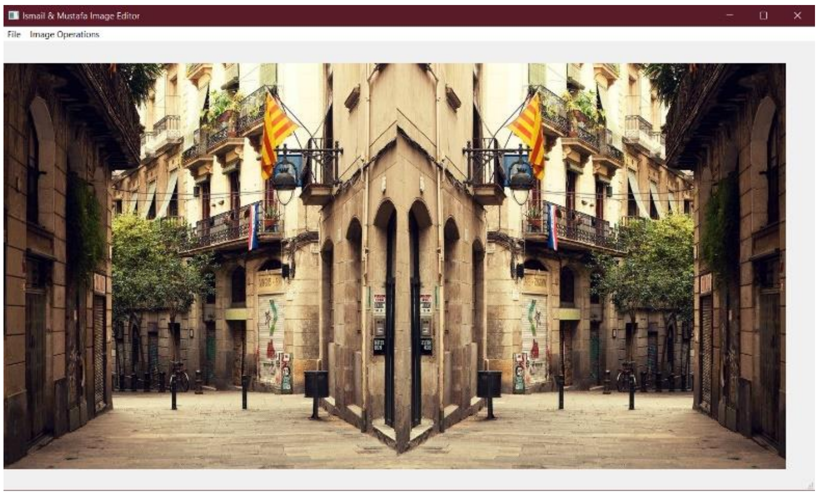


Photo-8

## 2.8 Mirror Image

This time, we get an image as if we put a mirror next to our original picture that we saw in Photo-8. We used the translation method of the OpenCV library as a filter. This method is used to translate a 2D array. The cv::flip function rotates a 2D array around vertical, horizontal, or both axes based on the input value we provide.

```
def mirrorImage(outputImageLabel, stateManager):
    if checkInputPhotoExist(stateManager):
        if stateManager.outputSource != '':
            inputImage = QImageToCvMat(stateManager.outputSource)
        else:
            inputImage = imread(stateManager.inputSource)

        outputData = opencv.flip(inputImage, 1)
        outputQImage = qimage2ndarray.array2qimage(outputData, normalize=True)
        outputImageLabel.setPixmap(QtGui.QPixmap(outputQImage))
        stateManager.imageOperation(
            outputQImage)
    else:
        warnMessage("Please open a image!")
```

## 2.9 Rotate Image

We can rotate our original picture, which we see in Photo-9, with an angle of 90 degrees counterclockwise. As I will talk about later, since our program works cumulatively, we get a 180-degree angle when the filter is applied again. We used the rotate method of the imutils library as a filter. With this method, we can rotate it as much as we want with the input we give the multiples of 90 degrees counterclockwise.

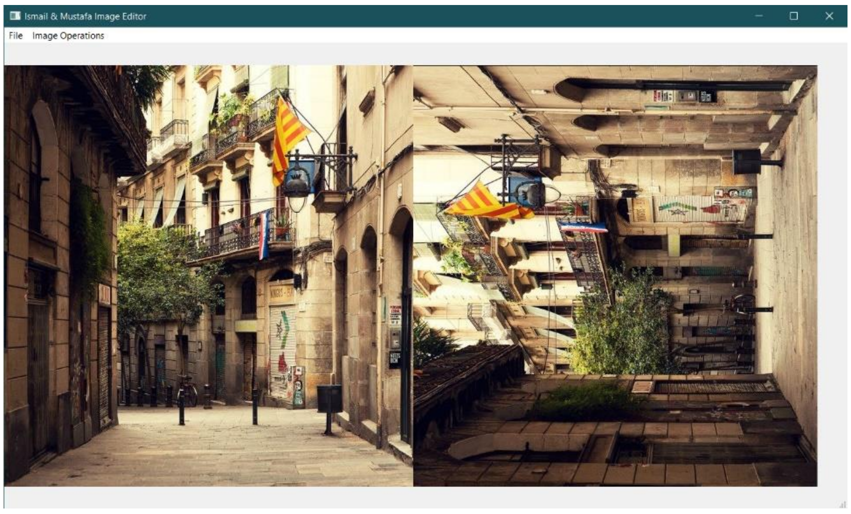


Photo-9

```
def rotateImage(outputImageLabel, stateManager):
    if checkInputPhotoExist(stateManager):
        if stateManager.outputSource != '':
            inputImage = QImageToCvMat(stateManager.outputSource)
        else:
            inputImage = imread(stateManager.inputSource)
        outputData = imutils.rotate(inputImage, 90)
        outputQImage = qimage2ndarray.array2qimage(outputData, normalize=True)
        outputImageLabel.setPixmap(QtGui.QPixmap(outputQImage))
        stateManager.imageOperation(
            outputQImage)
    else:
        warnMessage("Please open a image!")
```

## 2.10 Reverse Colours of Image

We reverse the colours in our original picture that we saw in Photo-10. We used the bitwise_not method of the OpenCV library as a filter. Inverting an image means reversing the colors on the image. For example, the inverted color for red color will be (0, 255, 255). Note that 0 became 255 and 255 became 0. This means that inverting an image is essentially subtracting the old RGB values from 255.

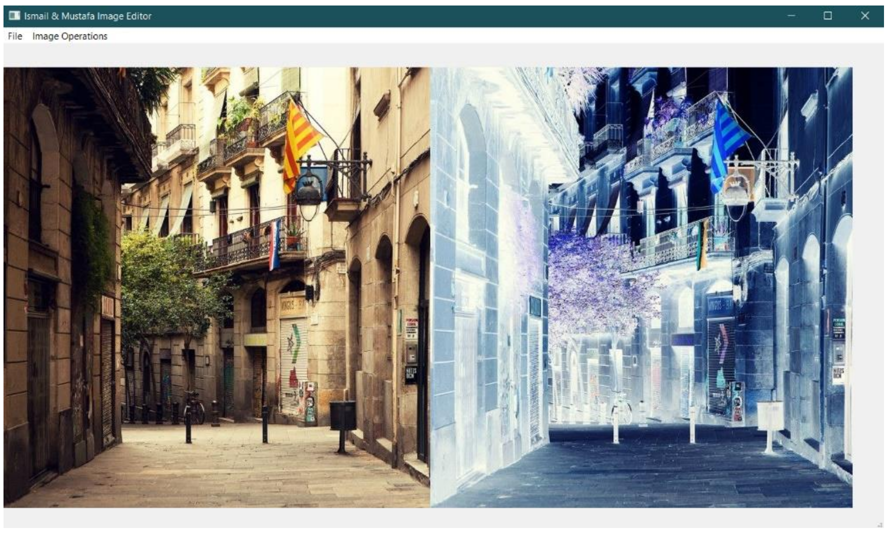


Photo-10

```
def reverseColor(outputImageLabel, stateManager):
    if checkInputPhotoExist(stateManager):
        if stateManager.outputSource != '':
            inputImage = QImageToCvMat(stateManager.outputSource)
        else:
            inputImage = imread(stateManager.inputSource)
        outputData = opencv.bitwise_not(inputImage)
        outputQImage = qimage2ndarray.array2qimage(outputData, normalize=True)
        outputImageLabel.setPixmap(QtGui.QPixmap(outputQImage))
        stateManager.imageOperation(
            outputQImage)
    else:
        warnMessage("Please open a image!")
```

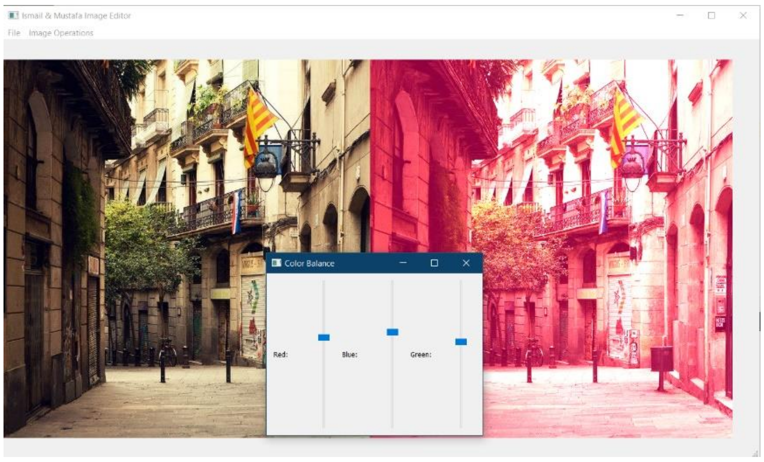


Photo-11

## 2.11 Colour Balance

By increasing or decreasing the Red, Blue, and Green (RGB) colours of our original picture that we saw in Photo-11, we obtain a new picture with the tones we want. We process the values we get from the slider as code directly by changing the channels on the relevant matrix. Since our code is long, we only put the photo of the part we are processing.

```
if value > 50:
    increaseSize = (255 // 50) * (value - 50)
    for i in range(inputImage.shape[0]):
        for j in range(inputImage.shape[1]):
            pixel = inputImage[i, j][channel]
            if pixel + increaseSize >= 255:
                outputImage[i, j][channel] = 255
            else:
                outputImage[i, j][channel] = pixel + increaseSize
if value < 50:
    decreaseSize = (255 // 50) * (50 - value)
    for i in range(inputImage.shape[0]):
        for j in range(inputImage.shape[1]):
            pixel = inputImage[i, j][channel]
            if pixel - decreaseSize <= 0:
                outputImage[i, j][channel] = 0
            else:
                outputImage[i, j][channel] = pixel - decreaseSize
```

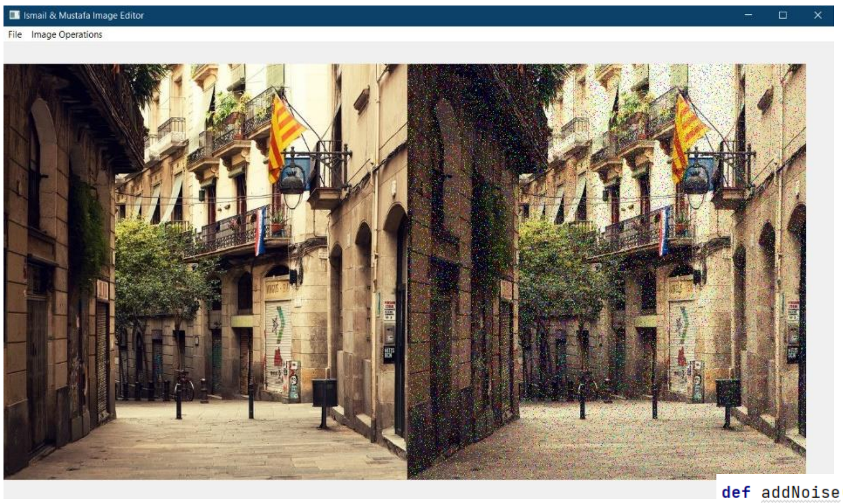


Photo-12

## 2.12 Add Noise

We add noise to our original picture that we saw in Photo-12. We used the random noise method of the scikit-image library as a filter. The method we use Replaces random pixels with either 1 or low_val, where low_val is 0 for unsigned images or -1 for signed images. In addition, with this method, you can add different noises such as pepper, Gaussian noise by giving different parameters. The reason why we did not use Pepper, or any other method was to learn a noise that we did not experience in the lesson.

```
def addNoise(outputImageLabel, stateManager):
    if checkInputPhotoExist(stateManager):
        if stateManager.outputSource != '':
            inputImage = QImageToCvMat(stateManager.outputSource)
        else:
            inputImage = imread(stateManager.inputSource)

        outputData = random_noise(inputImage, mode='s&p', amount=0.1)
        outputData = np.array(255 * outputData, dtype='uint8')
        outputQImage = qimage2ndarray.array2qimage(outputData, normalize=True)
        outputImageLabel.setPixmap(QtGui.QPixmap(outputQImage))
        stateManager.imageOperation(
            outputQImage)
    else:
        warnMessage("Please open a image!")
```

## 2.13 Adjust Brightness, Contrast, and Saturation of Image

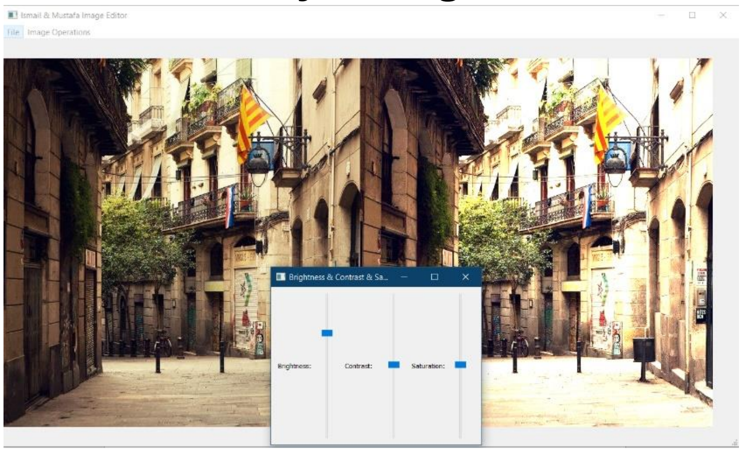


Photo-13

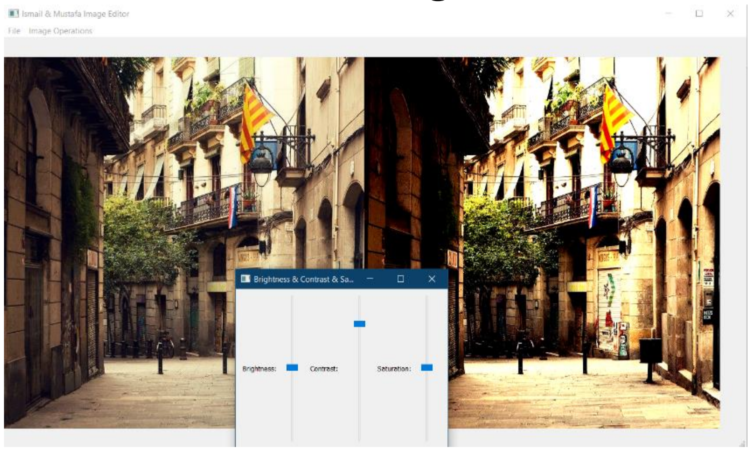


Photo-14

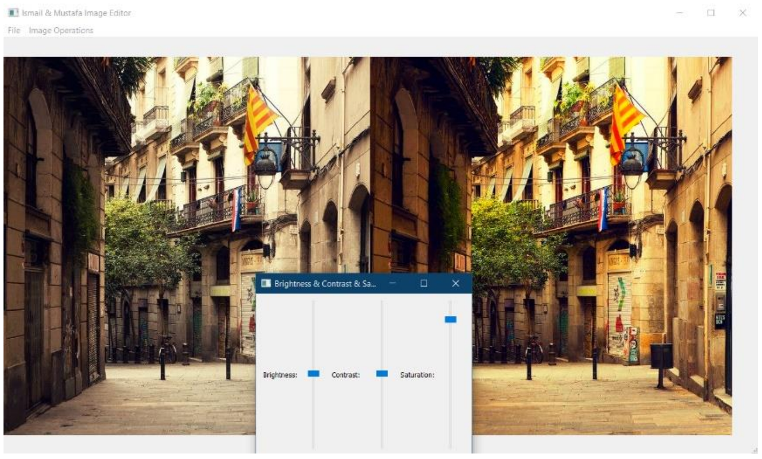
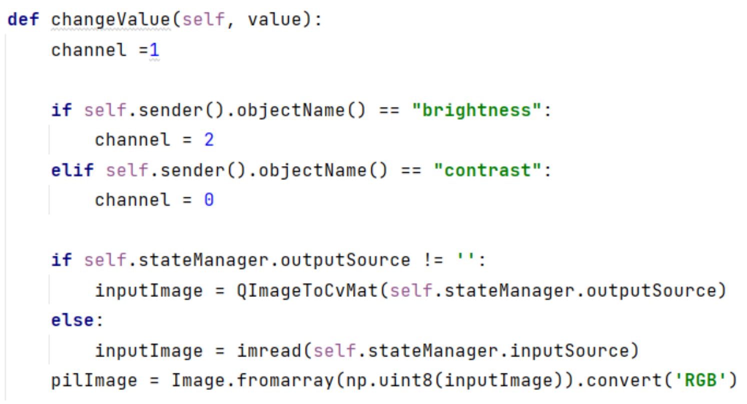


Photo-15

We add brightness to our original picture that we saw in Photo-13. We add a contrast to our original picture that we saw in Photo-13. We add saturation to our original picture that we saw in Photo-15. We used the enhance method of the PIL library as a filter. We preferred the method we use because it allows us to perform more than one operation by giving different parameters. Thus, we have handled 3 processes with a single library.

Here are the methods we use:

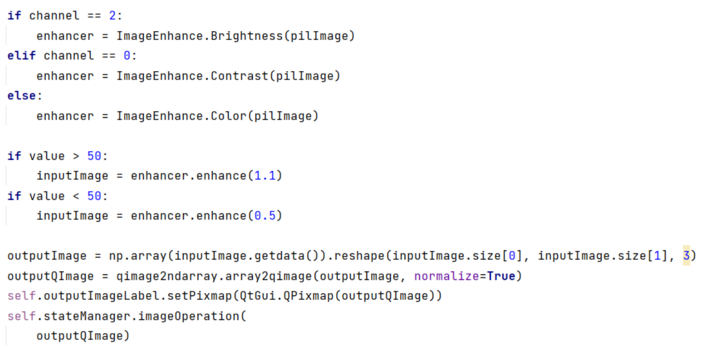
- This class can be used to control the brightness of an image. An enhancement factor of 0.0 gives a black image. A factor of 1.0 gives the original image. We change the factor with slide.
- This method can be used to control the contrast of an image, similar to the contrast control on a TV set. An enhancement factor of 0.0 gives a solid gray image. A factor of 1.0 gives the original image. We change the factor with slide.
- This method can be used to adjust the sharpness of an image. An enhancement factor of 0.0 gives a blurred image, a factor of 1.0 gives the original image, and a factor of 2.0 gives a sharpened image.



```
def changeValue(self, value):
    channel =1

    if self.sender().objectName() == "brightness":
        channel = 2
    elif self.sender().objectName() == "contrast":
        channel = 0

    if self.stateManager.outputSource != '':
        inputImage = QImageToCvMat(self.stateManager.outputSource)
    else:
        inputImage = imread(self.stateManager.inputSource)
    pilImage = Image.fromarray(np.uint8(inputImage)).convert('RGB')
```



```
if channel == 2:
    enhancer = ImageEnhance.Brightness(pilImage)
elif channel == 0:
    enhancer = ImageEnhance.Contrast(pilImage)
else:
    enhancer = ImageEnhance.Color(pilImage)

if value > 50:
    inputImage = enhancer.enhance(1.1)
if value < 50:
    inputImage = enhancer.enhance(0.5)

outputImage = np.array(inputImage.getdata()).reshape(inputImage.size[0], inputImage.size[1], 3)
outputQImage = qimage2ndarray.array2qimage(outputImage, normalize=True)
self.outputImageLabel.setPixmap(QtGui.QPixmap(outputQImage))
self.stateManager.imageOperation(
    outputQImage)
```

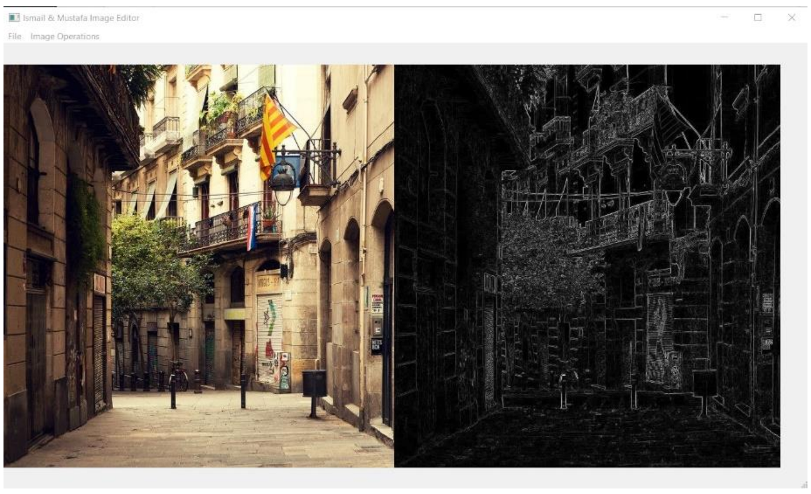
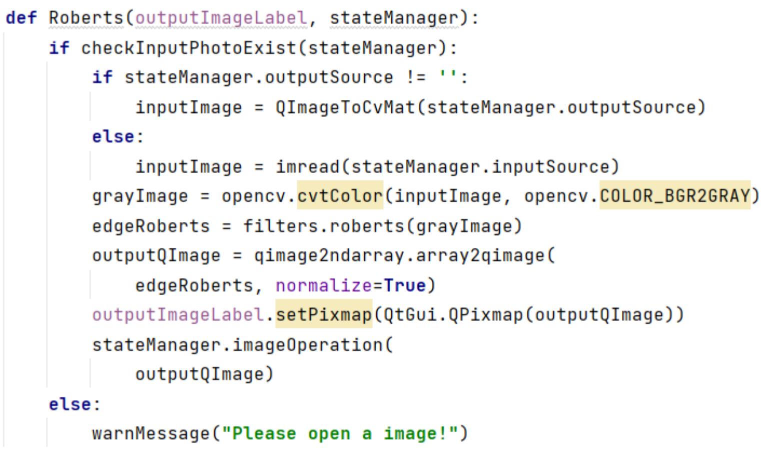


Photo-16



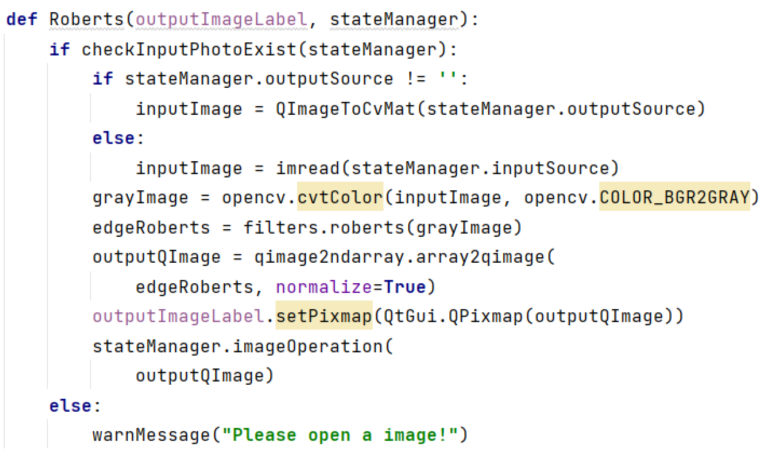
```python
def Roberts(outputImageLabel, stateManager):
    if checkInputPhotoExist(stateManager):
        if stateManager.outputSource != '':
            inputImage = QImageToCvMat(stateManager.outputSource)
        else:
            inputImage = imread(stateManager.inputSource)
        grayImage = opencv.cvtColor(inputImage, opencv.COLOR_BGR2GRAY)
        edgeRoberts = filters.roberts(grayImage)
        outputQImage = qimage2ndarray.array2qimage(
            edgeRoberts, normalize=True)
        outputImageLabel.setPixmap(QtGui.QPixmap(outputQImage))
        stateManager.imageOperation(
            outputQImage)
    else:
        warnMessage("Please open a image!")
```

## 2.14 Detect Edges of Image

We apply a filter that shows the edges of our original picture that we see in Photo-16. We used the Roberts method of the OpenCV library as a filter. In the method we use, we first apply the grayscale filter (the same method we use with 2.5 Grayscale) to our image. Then we apply the Robert filter. The method uses the difference between two adjacent pixels in the diagonal direction to approximate the gradient amplitude to detect edges. The effect of detecting vertical edges is better than that of oblique edges, the positioning accuracy is high, and it is sensitive to noise and cannot suppress the influence of noise. We chose this filter because our image is free of noise and slashes.

# 3 How Our Program Work?

## 3.1 How Our Image Operation Work?



```python
def Roberts(outputImageLabel, stateManager):
    if checkInputPhotoExist(stateManager):
        if stateManager.outputSource != '':
            inputImage = QImageToCvMat(stateManager.outputSource)
        else:
            inputImage = imread(stateManager.inputSource)
        grayImage = opencv.cvtColor(inputImage, opencv.COLOR_BGR2GRAY)
        edgeRoberts = filters.roberts(grayImage)
        outputQImage = qimage2ndarray.array2qimage(
            edgeRoberts, normalize=True)
        outputImageLabel.setPixmap(QtGui.QPixmap(outputQImage))
        stateManager.imageOperation(
            outputQImage)
    else:
        warnMessage("Please open a image!")
```

Photo-17

I will briefly explain how Image Operations work through the code example we saw in Photo-17.

**3.1.1.** We are checking if there is any opened image file.

**3.1.2.** If there is no image opened, a warn pop-up appears on the screen.

**3.1.3.** If we have an input image, we convert our image to a NumPy array.

**3.1.4.** Since our program works cumulatively, if there is a processed input image, we include it in the process instead of the first 3 steps.

**3.1.5.** When we have a photograph to be processed, we carry out our operations in order.

**3.1.6.** As in the code example, we apply grayscale to our image, respectively.

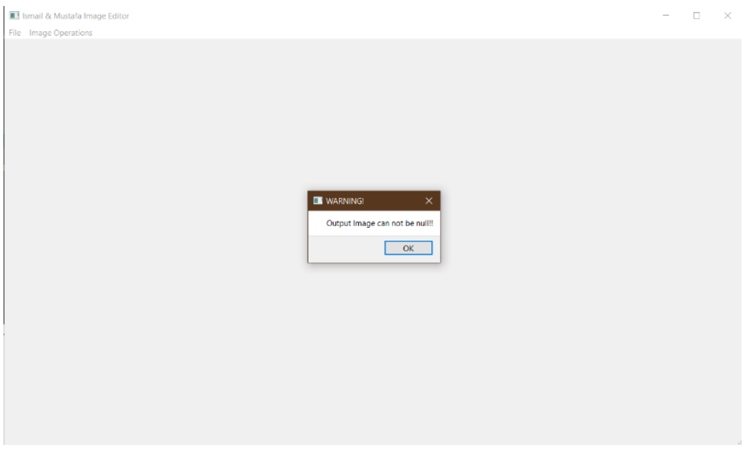
**3.1.7.** We apply the Roberts filter.

**3.1.8.** When we are done, we convert our NumPy array to q image format.

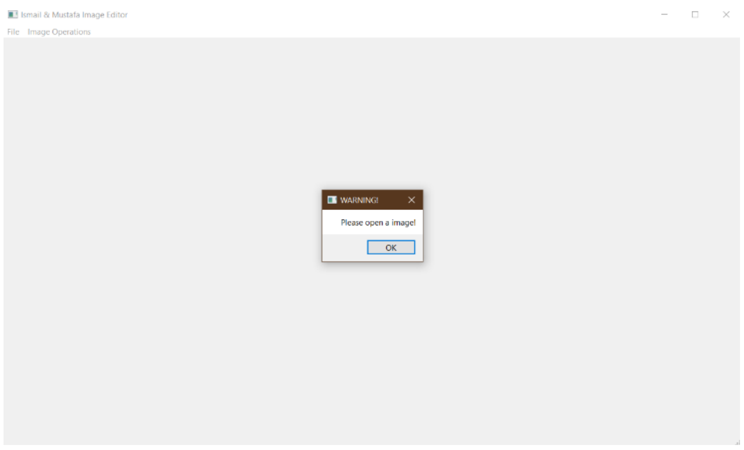
**3.1.9.** We update the "outputimagelayout" so that our processed function appears on the screen.

**3.1.10.** We send the output to the statemanager for the next process.

## 3.2 What Are Our Classes For?



Output Image can not be null!!



Please open a image!

Photo-18

Photo-19

The classes in our program are as follows:

**3.2.1.** Our main.py class is our main class, and it performs the necessary initialization works (Gui setup).

**3.2.2.** Our ui.py class includes many processes such as:

- Initialization of statemanager, labels, actions, layouts (vertical, horizontal)
- Actions definitions when they clicked a triggered action ex: self.actionReverse_Image.triggered.connect(lambda: Operations.reverseColor(self.outputImage, stateManager))
- Runs some stateManager operations. For example, check state, outputImage (QImage), inputImage(Path).
- Output Image layout for update the layout.

**3.2.3.** Our StateManager.py class ensures that the input image operations of our program are run properly. A required class for it to work cumulatively.

**3.2.4.** Our Operation.py class performs the following operations:

- Converting our Q image to a NumPy array
- If there is an "Error", display warn pop-up. For example, if we do not have a processed image, if the user calls the save image function, he will encounter an error message (Photo-18 & Photo-19).

## 4 Cumulative

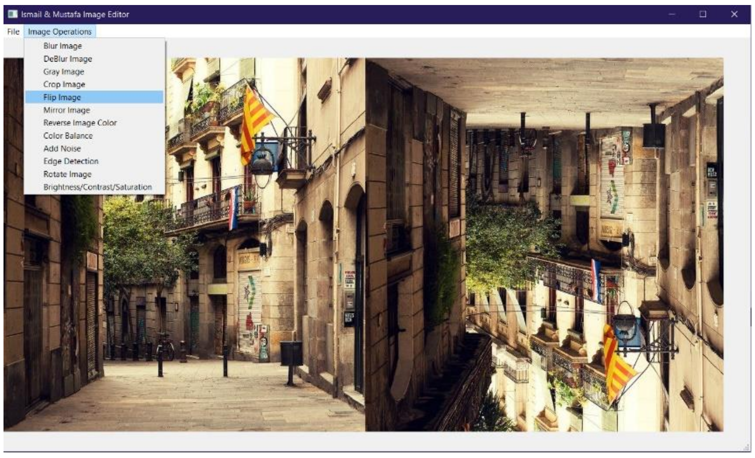


Photo-20

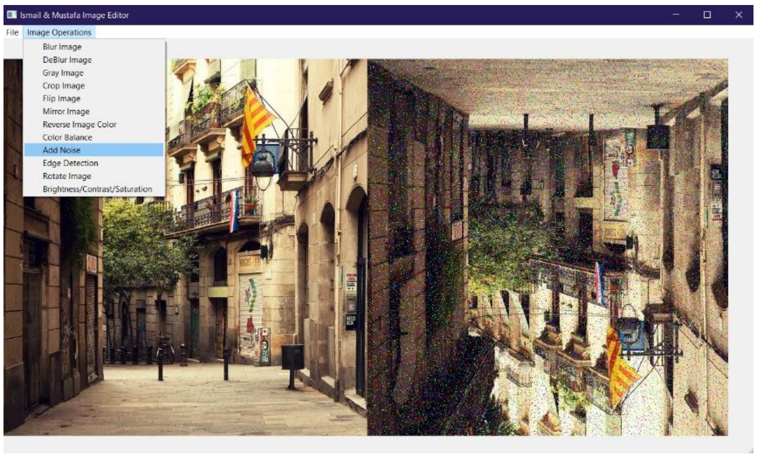


Photo-21

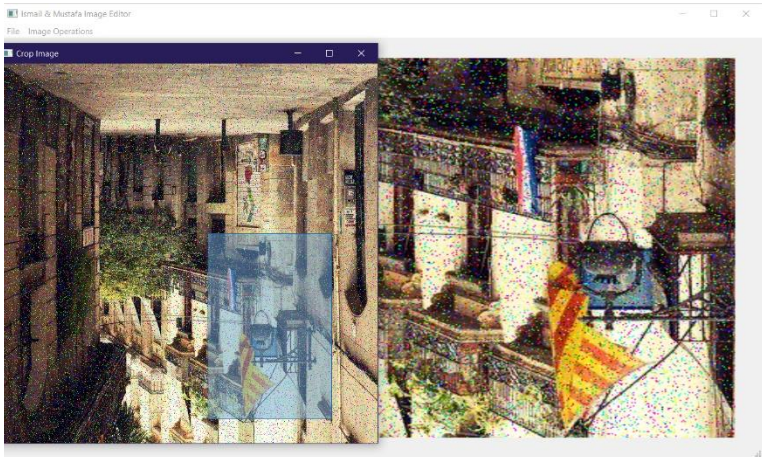


Photo-22

As we can see in Photo-20, Photo-21, and Photo-22, our image operators work cumulatively. All situations are considered and accordingly, extensions are observed in our image operators and are checked beforehand. In other words, all of our functions can run one after the other without any problems.

## 5 Lessons Learned from the Project

First of all, during this project, we learned how to use what we learned during our Computer Image course in a real project. In addition, while doing research on the project, we learned new filters apart from the course content and reinforced the filters we saw in the course. In addition, we have gained desktop application writing experience thanks to the QT framework. Thanks to the project, we realized that image filters also have different requirements. For example, we need to Grayscale our image before applying the Robberts filter. In short, we had good experiences with Computer Image thanks to this project.