

Desktop Image Processing Application.

Submitted to:

Dr. Rabab Farouk,

TA. Afaf Saad.

Computer Engineering Department.

Module: Digital Image Processing.

Module Code: 22COMP16H.

Prepared by:

Hossam Hassan (180871)

Mirna Victor (190860)

Jonathan Joseph (189059)

Year Four Computer Engineering.

Date Submitted: 3 May 2023.

Table of Contents

1. Introduction 1

2. Application Implementation 3

2.1 Requirements:..... 3

2.2 Features 4

1.1 Draw a Circle:5

1.2 Draw a rectangle:.....5

1.3 Brightness:.....6

1.4 Put Text:8

1.5 Resize:9

1.6 Thresholding:11

1.7 Histogram:.....12

1.8 Edge detection:.....13

1.9 Feature matching:.....14

A Guide to Using a Desktop Application: 16

Conclusion: 17

Table of Figures

Figure 1 - Python + PyQt5 3

Figure 2 - requirements.txt file..... 3

Figure 3 - Main Menu 4

Figure 4 - Draw_circle_function..... 5

Figure 5 - Draw_rectangle_function 6

Figure 6 - Gamma_correction_brightness..... 6

Figure 7 - Adjust_gamma_value..... 7

Figure 8 - Gamma correction with the adjusted value 7

Figure 9 - put_text..... 8

Figure 10 - Steps of put_text function..... 9

Figure 11 - Resize function. 9

Figure 12 - Resize function steps. 10

Figure 13 - Thresholding function 11

Figure 14 - Histogram function..... 12

Figure 15 - Select gradient technique..... 13

Figure 16 - Result by Canny edge detector 13

Figure 17 - Feature matching function using sift algorithm..... 15

1. Introduction

Digital Image Processing (DIP) is a dynamic field that deals with the manipulation and analysis of digital images using computer algorithms. As the availability of digital cameras, smartphones, and other imaging devices has increased, the need for efficient and effective ways to process and analyze images has become more critical than ever before. DIP plays a vital role in many domains, including medicine, remote sensing, robotics, and security.

DIP involves a range of techniques, from basic operations like image enhancement and restoration to advanced algorithms such as pattern recognition and machine learning. As a result, it has become an essential tool for researchers, engineers, and practitioners in various fields who seek to extract meaningful information from images. The primary aim of this introduction is to provide a brief overview of the fundamental concepts and applications of DIP, along with some of the challenges and future directions in the field.

The process of image processing typically involves several stages, including acquisition, pre-processing, segmentation, feature extraction, and classification. Each stage plays a crucial role in producing accurate and reliable results. Image acquisition is the process of capturing an image using a digital camera or other imaging device. Pre-processing techniques are then used to clean up the image, removing noise and artifacts and correcting any distortion or errors introduced during the acquisition process.

Segmentation is the process of dividing an image into meaningful regions or objects. This is typically done using various algorithms that identify edges, boundaries, and other features of interest. Feature extraction involves identifying and quantifying the characteristics of the segmented regions, such as color, texture, and shape.

Classification is the final stage of image processing, where the segmented regions are classified into different categories based on their features. This is done using various classification algorithms, such as support vector machines, decision trees, or neural networks.

DIP has numerous applications, including medical imaging, satellite and aerial imaging, industrial inspection, and surveillance. In medicine, DIP is used for tasks such as identifying tumors, detecting bone fractures, and analyzing brain function. In satellite and aerial imaging, DIP is used to analyze terrain, vegetation, and weather patterns. In industrial inspection, DIP is used to detect defects in manufactured products, such as cracks in metal parts. In surveillance, DIP is used for tasks such as

face recognition, object tracking, and motion detection.

Digital Image Processing is a rapidly evolving field that deals with the manipulation and analysis of digital images using computer algorithms. With the increasing availability of digital cameras, smartphones, and other imaging devices, the need for efficient and effective ways to process and analyze images has become more important than ever before. In response to this growing demand, we have decided to design a user-friendly application that enables users to edit images with ease. This digital image processing application will incorporate various techniques and algorithms to perform basic and advanced image processing operations.

The application will be developed using Python programming language and will utilize popular libraries such as PyQt5, Numpy, Matplotlib, Seaborn, and OpenCV. Our goal is to provide an intuitive interface that will allow users to access various features and tools easily. The application will include features such as opening and resizing images, converting images to grayscale, inverting them, applying thresholding, blurring, or gradient effects, calculating a histogram, drawing circles or rectangles, performing morphological operations, blending images, and more.

The application we are designing aims to provide a comprehensive image processing solution that caters to the needs of professionals in various domains. With this application, users will be able to perform image processing tasks with ease and efficiency, thereby enhancing their productivity and effectiveness. Our goal is to develop a user-friendly and intuitive digital image processing application that will enable users to process and analyze images efficiently and accurately, thereby facilitating advancements in various fields.

In recent years, DIP has seen significant advancements due to the emergence of new technologies such as deep learning and computer vision. These advancements have enabled more accurate and efficient image processing and analysis, leading to new applications and opportunities in various fields. Therefore, DIP has become an exciting and rapidly evolving field with vast potential for innovation and growth.

2. Application Implementation

This is a user-friendly desktop application that has been designed to facilitate the application of an array of sophisticated image processing techniques to images. Built using the popular Python programming language, the application's graphical user interface (GUI) has been developed using PyQt5, a powerful GUI toolkit. The image processing functionalities of the application have been implemented using OpenCV, a free, open-source computer vision and machine learning software library. The application's user interface is intuitive and allows for easy interaction, enabling users to seamlessly select and apply various image processing operations to their images. The application is an excellent tool for researchers, professionals, and hobbyists who want to enhance, analyze, and transform their digital images in a fast and efficient manner.

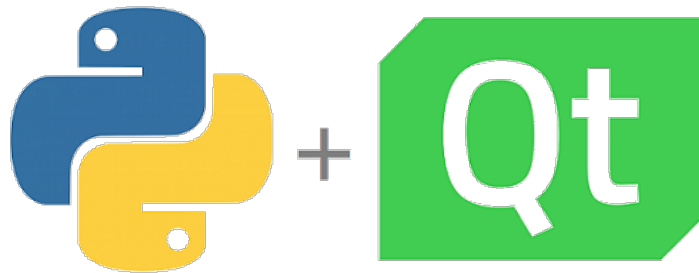


Figure 1 - Python + PyQt5

2.1 Requirements:

In order to use this application, a set of software packages must be installed on the user's computer beforehand. These packages include Python 3.x, PyQt5, Numpy, Matplotlib, Seaborn, and OpenCV. Python 3.x is the programming language in which the application has been developed and is a widely used and versatile language for software development. PyQt5 is a powerful GUI toolkit for developing cross-platform desktop applications, and it provides a rich set of components for creating modern graphical user interfaces. Numpy is a Python library that is used for working with arrays, and it is a fundamental package for scientific computing with Python. Matplotlib is a plotting library for Python that provides an array of visualization tools for displaying data and creating graphical presentations. Seaborn is a Python data visualization library that builds on top of Matplotlib and provides an additional layer of visual appeal to plots and graphs. Finally, OpenCV is a popular computer vision and machine learning library that is used for image and video processing. By requiring these packages, the application provides a comprehensive and powerful set of tools for users to perform image processing operations with ease and efficiency.

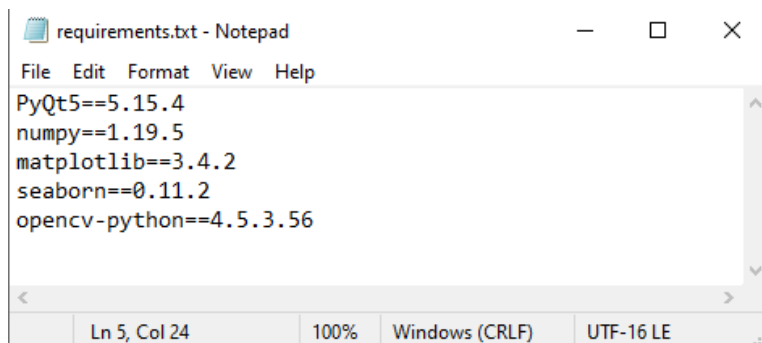
A screenshot of a Notepad window titled 'requirements.txt - Notepad'. The window contains the following text: PyQt5==5.15.4, numpy==1.19.5, matplotlib==3.4.2, seaborn==0.11.2, and opencv-python==4.5.3.56. The status bar at the bottom shows 'Ln 5, Col 24', '100%', 'Windows (CRLF)', and 'UTF-16 LE'.

Figure 2 - requirements.txt file

2.2 Features

This application offers a multitude of diverse image processing features that can be used to enhance and transform digital images in a variety of ways. The extensive range of capabilities provided by this application includes features such as opening and resizing images, converting them to grayscale, inverting them, applying thresholding, blurring, or gradient effects, calculating a histogram, drawing circles or rectangles, performing morphological operations, blending images, and much more. These features have been designed to offer users a comprehensive suite of image processing tools that can be applied to meet their specific needs, regardless of whether they are using the application for personal or professional use. The application's intuitive and user-friendly interface ensures that users can effortlessly access these features and apply them to their images in a matter of seconds. With its robust and flexible image processing capabilities, this application is an indispensable tool for anyone seeking to effortlessly and efficiently enhance and manipulate digital images.

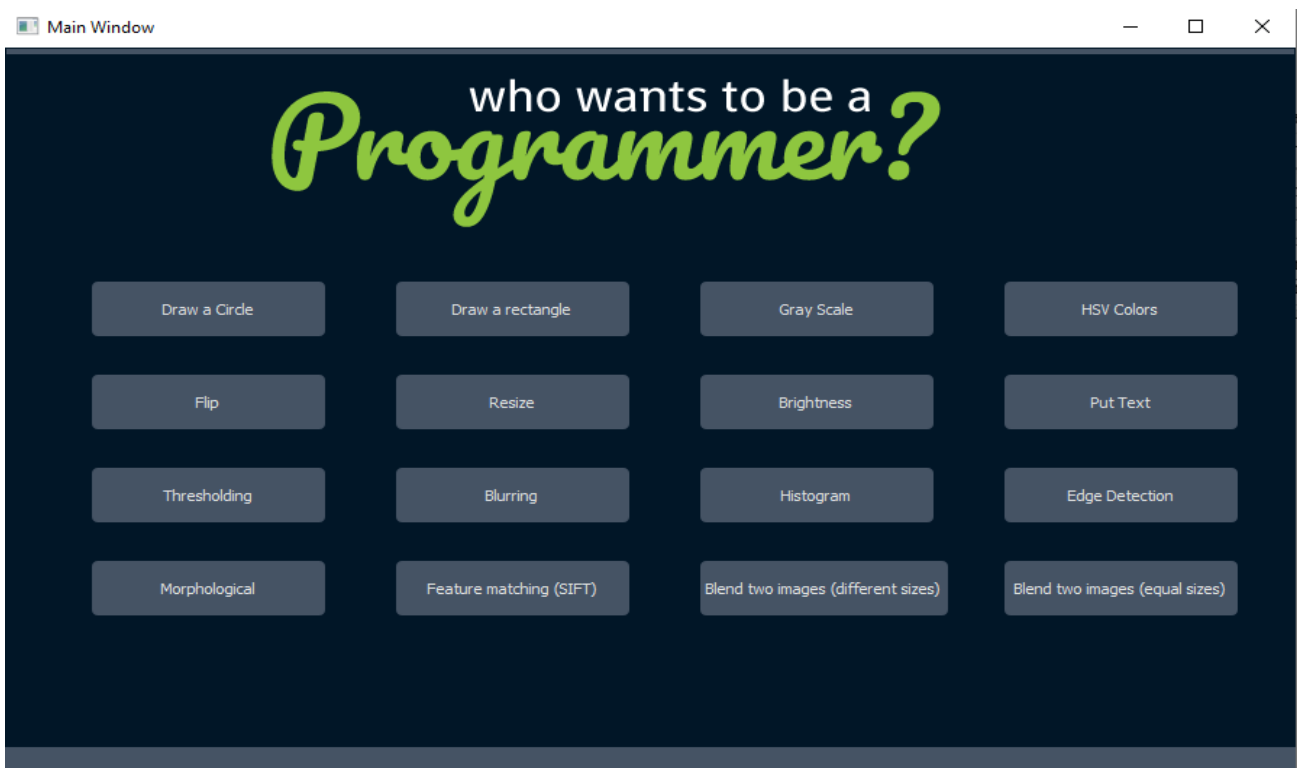


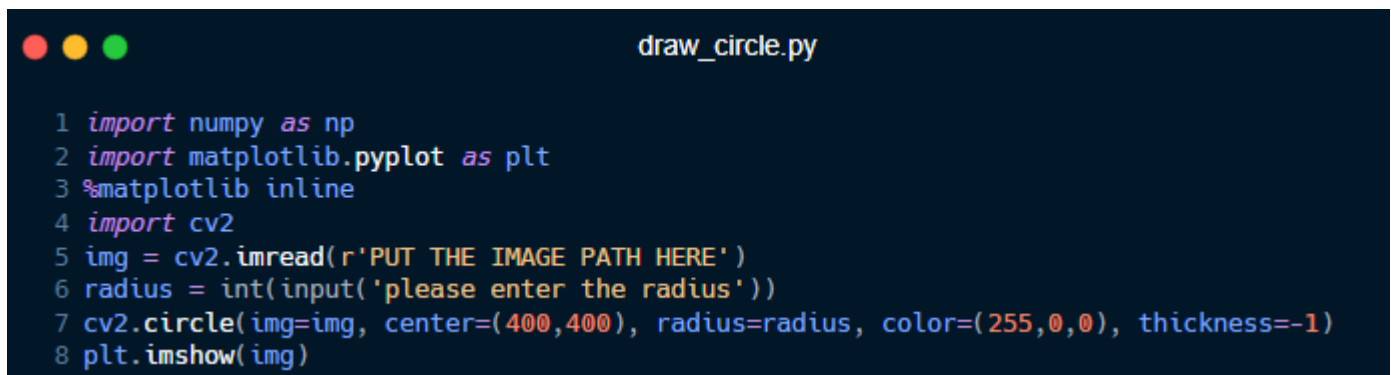
Figure 3 - Main Menu

1.1 Draw a Circle:

This code imports the required libraries and reads an image from a specified path using the OpenCV library in Python. It then prompts the user to enter the radius of the circle they want to draw on the image.

After the user enters the radius, the code uses the `cv2.circle` function to draw a circle on the image with the given center coordinates and radius. The color argument specifies the color of the circle as a tuple of RGB values. The thickness argument is set to `-1`, which means that the circle will be filled with the specified color.

Finally, the modified image is displayed using `plt.imshow`. Note that the circle is drawn directly on the original image (`img`), so the modified image is displayed instead of a new blank image.



```
draw_circle.py

1 import numpy as np
2 import matplotlib.pyplot as plt
3 %matplotlib inline
4 import cv2
5 img = cv2.imread(r'PUT THE IMAGE PATH HERE')
6 radius = int(input('please enter the radius'))
7 cv2.circle(img=img, center=(400,400), radius=radius, color=(255,0,0), thickness=-1)
8 plt.imshow(img)
```

Figure 4 - Draw_circle_function

1.2 Draw a rectangle:

This is a code snippet written in Python that makes use of several libraries to perform an image processing operation.

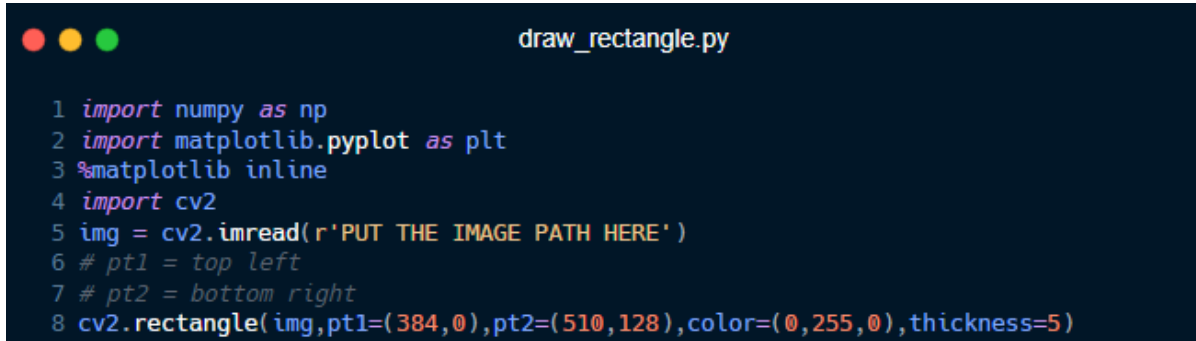
First, the `numpy` and `matplotlib` libraries are imported, which are commonly used for scientific computing and plotting. The `%matplotlib inline` command is used to display the plotted images within the Jupyter notebook or IPython console.

Then, the `cv2` library, which stands for OpenCV (Open Source Computer Vision), is imported. OpenCV is a popular open-source library that provides tools and functions for computer vision and image processing.

Next, an image is loaded using the `cv2.imread()` function. The function takes in the path of the image file to be loaded and returns a NumPy array representing the image.

After that, a rectangle is drawn on the image using the `cv2.rectangle()` function. The function takes in the image as the first argument and several other parameters, including the coordinates of the top left and bottom right corners of the rectangle (`pt1` and `pt2` respectively), the color of the rectangle (in RGB format), and the thickness of the lines used to draw the rectangle.

In this case, the rectangle is drawn from the coordinates (384,0) to (510,128) with a green color (represented by the tuple (0,255,0)) and a thickness of 5 pixels. The resulting image with the rectangle drawn on it can then be displayed using the matplotlib.pyplot.imshow() function.

A screenshot of a code editor window titled "draw_rectangle.py". The code is as follows:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 %matplotlib inline
4 import cv2
5 img = cv2.imread(r'PUT THE IMAGE PATH HERE')
6 # pt1 = top left
7 # pt2 = bottom right
8 cv2.rectangle(img,pt1=(384,0),pt2=(510,128),color=(0,255,0),thickness=5)
```

Figure 5 - Draw_rectangle_function

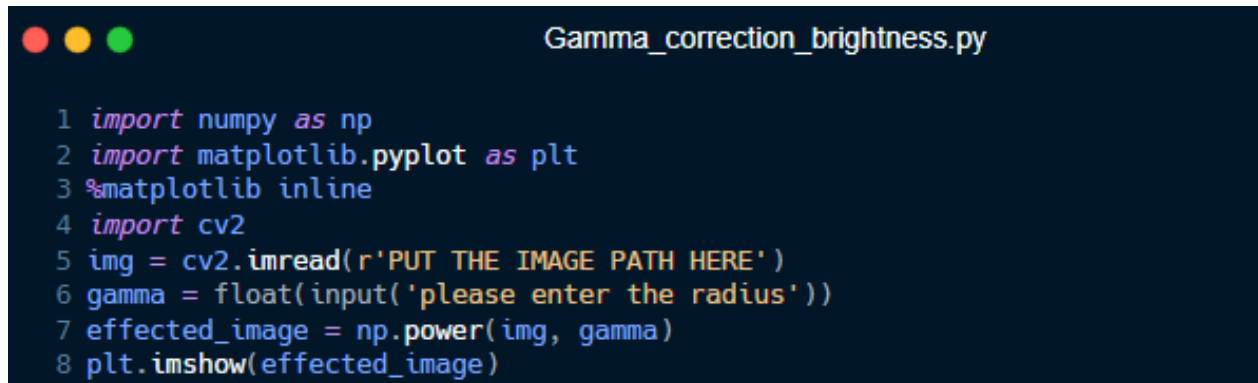
1.3 Brightness:

This code imports the necessary libraries and modules for image processing and displays. It then reads an image from a file path using the OpenCV (cv2) library's "imread" function.

The code asks the user to input a value for the "gamma" variable using the "input" function. Gamma is a parameter that can be used to adjust the brightness and contrast of an image.

The "effected_image" variable is created using NumPy's "power" function, which raises each pixel value in the image to the power of the gamma value. This results in an image with a different brightness and contrast than the original.

Finally, the "imshow" function from Matplotlib is used to display the processed image.

A screenshot of a code editor window titled "Gamma_correction_brightness.py". The code is as follows:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 %matplotlib inline
4 import cv2
5 img = cv2.imread(r'PUT THE IMAGE PATH HERE')
6 gamma = float(input('please enter the radius'))
7 effected_image = np.power(img, gamma)
8 plt.imshow(effected_image)
```

Figure 6 - Gamma_correction_brightness

We define a method brightness_function that opens a dialog box for adjusting the gamma value of an image file selected by the user. First, the method opens a file dialog to allow the user to select an image file. Then it reads the image file using OpenCV's cv2.imread function and converts the image color space from BGR to RGB using cv2.cvtColor.

Next, the method creates a horizontal slider with a range of 1 to 10 and an initial value of 0.4, and a label that displays the current value of the slider. It also creates a spinbox with a range of 0.01 to 10.00 and an initial value of 0.4, and connects it to a function that updates the slider value. The slider is also connected to a function that updates the label.

The slider, label, and spinbox are added to a layout and displayed in a dialog box. The user can adjust the gamma value by moving the slider or entering a value in the spinbox.

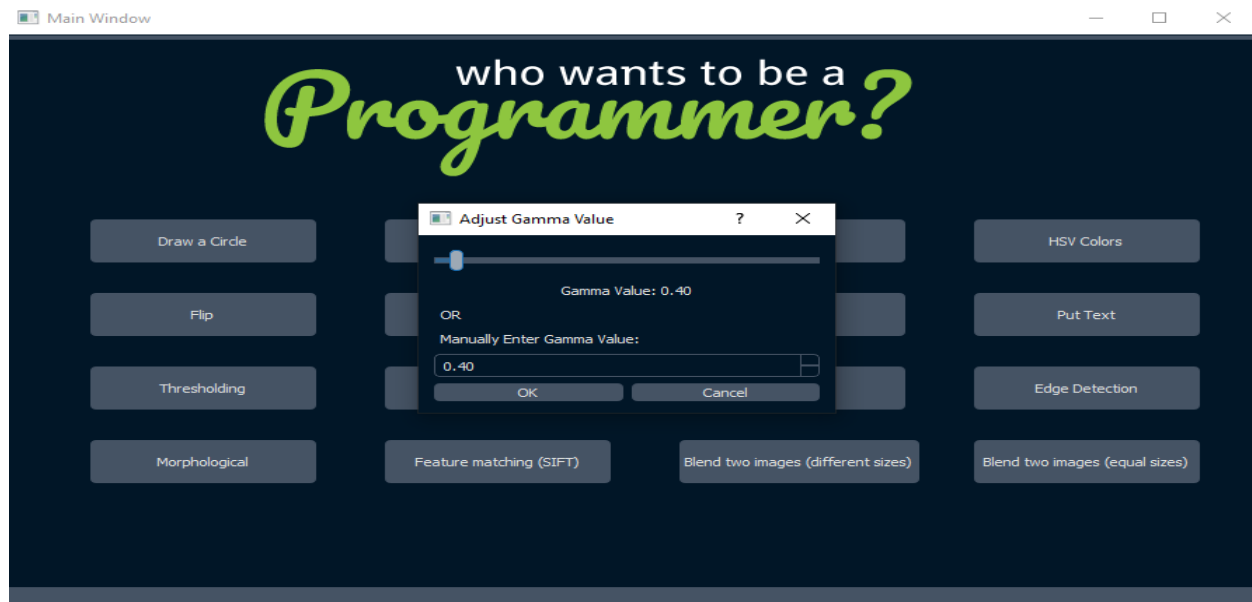
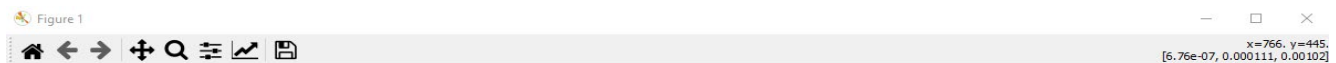


Figure 7 - Adjust_gamma_value



Gamma Correction with value 7.36

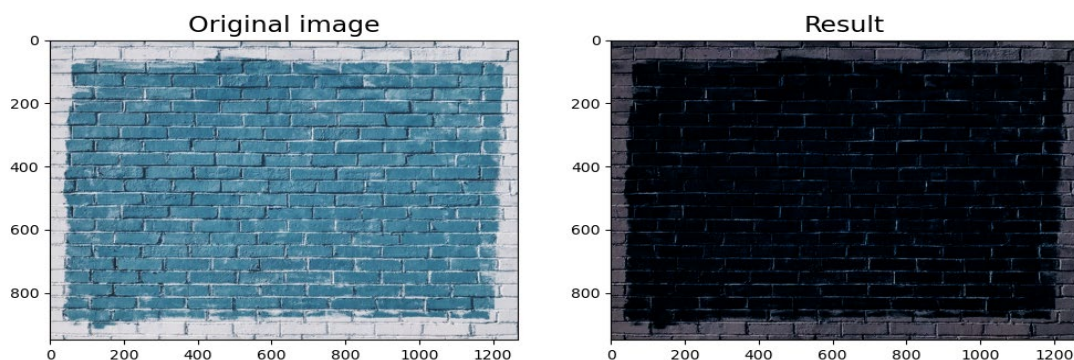


Figure 8 - Gamma correction with the adjusted value

1.4 Put Text:

This code reads an image file using OpenCV's `imread` function, and stores the resulting image as a NumPy array in the `img` variable. It then prompts the user to input a string of text using the `input` function, and stores the text in the `text` variable.

Next, the code specifies the font type to use for the text using the `cv2.FONT_HERSHEY_COMPLEX` constant. It then draws the specified text on the image using the `cv2.putText` function, passing in the image, text, starting point of the text, font type, font scale, text color, and thickness as arguments.

Finally, the code displays the modified image using matplotlib's `imshow` function. The resulting image will have the specified text drawn on it at the specified location.



```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 %matplotlib inline
4 import cv2
5 img = cv2.imread(r'PUT THE IMAGE PATH HERE')
6 font = cv2.FONT_HERSHEY_COMPLEX
7 text = input('please enter a string text')
8 cv2.putText(img, text='text', org=(10,600), fontFace=font, fontScale= 10, color=(255,0,0), thickness=4)
9 plt.imshow(img)
```

Figure 9 - `put_text`

To begin with, the function starts by prompting the user to select an image file using the `QFileDialog.getOpenFileName` method. Once the user selects an image, the file is loaded using the `cv2.imread` method, and its color space is converted from BGR (the default color space used by OpenCV) to RGB using the `cv2.cvtColor` method.

Next, the function displays the image in a Matplotlib plot using the `plt.imshow` method. The x-axis and y-axis are also labeled using the `plt.xlabel` and `plt.ylabel` methods, respectively. The ticks on the x and y axes are set using the `plt.xticks` and `plt.yticks` methods, and the `block` argument of the `plt.show` method is set to `False` so that the function doesn't wait for the user to close the window with the x-axis and y-axis.

To ensure that the function waits for the user to input text, font, color, size, and position before continuing, it uses a while loop that continuously processes events using the `QApplication.processEvents` method. To prevent the while loop from consuming too much CPU time, a `QTimer` with a timeout of 100 milliseconds is used to limit the number of times the loop is executed.

Once the user inputs the required values, the function creates a copy of the original image using the `img.copy` method and uses the `cv2.putText` method to add the text to the image. The modified image is then displayed in a new Matplotlib plot alongside the original image using the `plt.subplot` method.

Finally, the function uses the plt.show method to display the plot to the user.

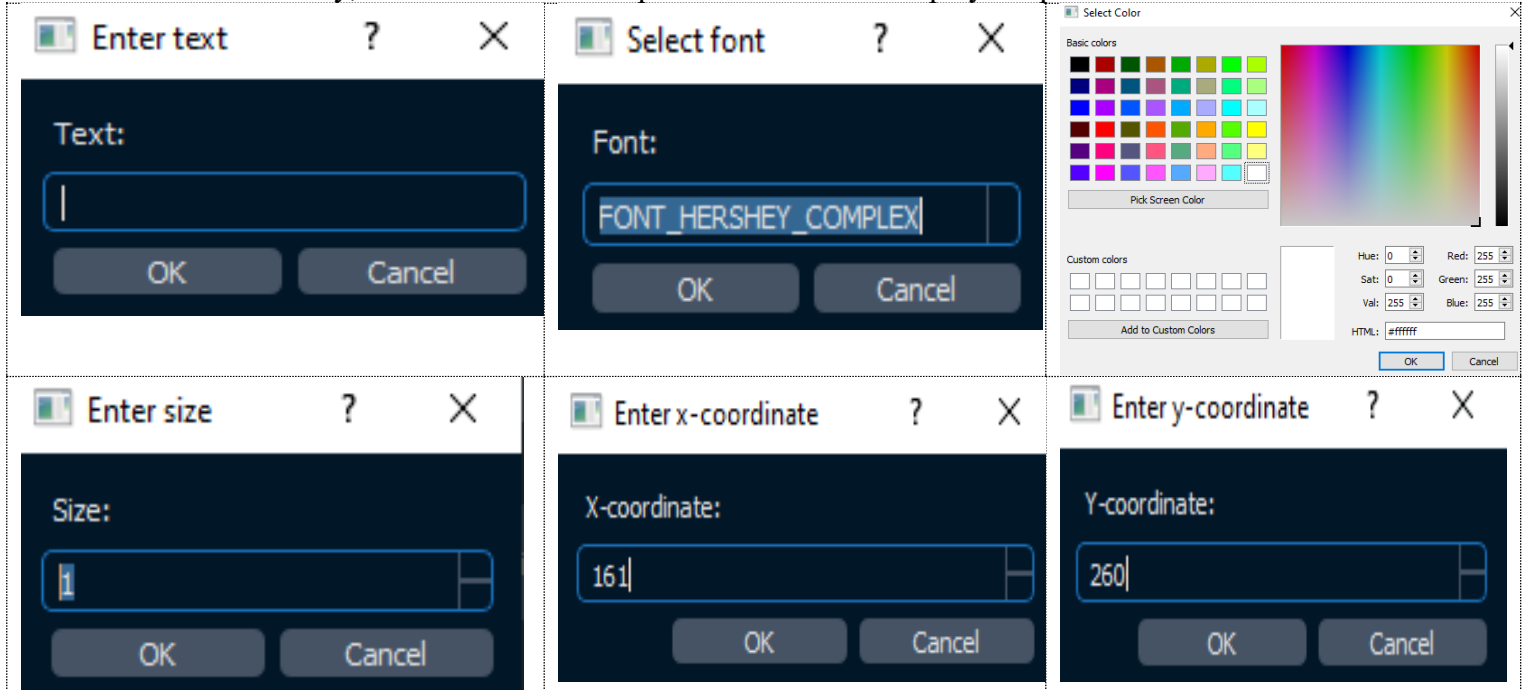


Figure 10 - Steps of put_text function.

1.5 Resize:

This code imports the necessary libraries numpy, matplotlib, and cv2 for image processing. Then it reads an image file located at the specified file path using cv2.imread function and converts its color space from BGR to RGB format using cv2.cvtColor function.

Next, the user is prompted to enter the desired width and height values for resizing the image using the cv2.resize function, and the resulting resized image is saved to a variable named resized_img.

Finally, the resized image is displayed using the plt.imshow function in a Matplotlib plot.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 %matplotlib inline
4 import cv2
5 img = cv2.imread(r'PUT THE IMAGE PATH HERE')
6 img = (cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
7 width = int(input('enter the width please'))
8 height = int(input('enter the height please'))
9 resized_img = cv2.resize(img, (width, height))
10 plt.imshow(resized_img)

```

Figure 11 - Resize function.

Our Python function that resizes an image specified by its file path. The function takes three parameters: self, width, and height. The self parameter is used to refer to the instance of the class to which the function belongs. The width and height parameters specify the new dimensions of the image to be resized.

First, the function loads the image from the specified file path using the OpenCV imread function, and then converts the color space of the image from BGR to RGB using the cvtColor function. Next, the function resizes the image using the OpenCV resize function with the specified width and height. After the image has been resized, the function creates a new figure with two subplots using the subplots function from the Matplotlib library. The original image is displayed in the first subplot, and the resized image is displayed in the second subplot. Each subplot is given a title using the set_title function, and the axes are turned off using the axis function. Finally, the tight_layout function is used to adjust the layout of the subplots and show function is used to display the plot.

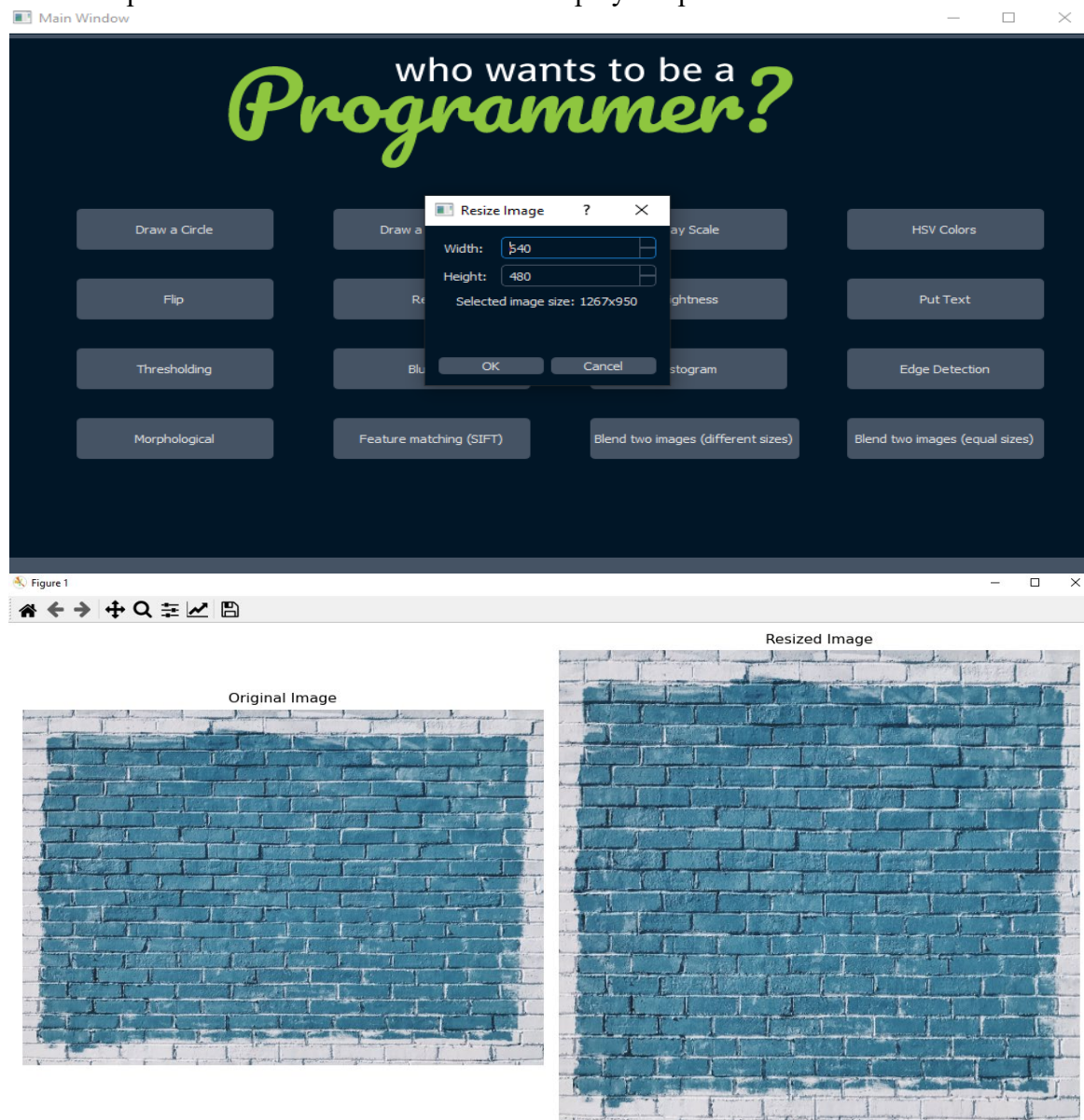


Figure 12 - Resize function steps.

1.6 Thresholding:

This code imports necessary libraries for image processing such as NumPy, Matplotlib, and OpenCV. It then reads an image file located at the specified path in grayscale mode using the `cv2.imread()` function and the flag `0`.

The user is prompted to enter a threshold value using the `input()` function. This threshold value will be used to create a binary image where pixel values above the threshold will be set to white (255) and pixel values below the threshold will be set to black (0).

The `cv2.threshold()` function is used to create the binary image with the specified threshold value. The `ret` variable stores the threshold value used by the function.

Finally, the binary image is displayed using `plt.imshow()` function with the 'gray' colormap.

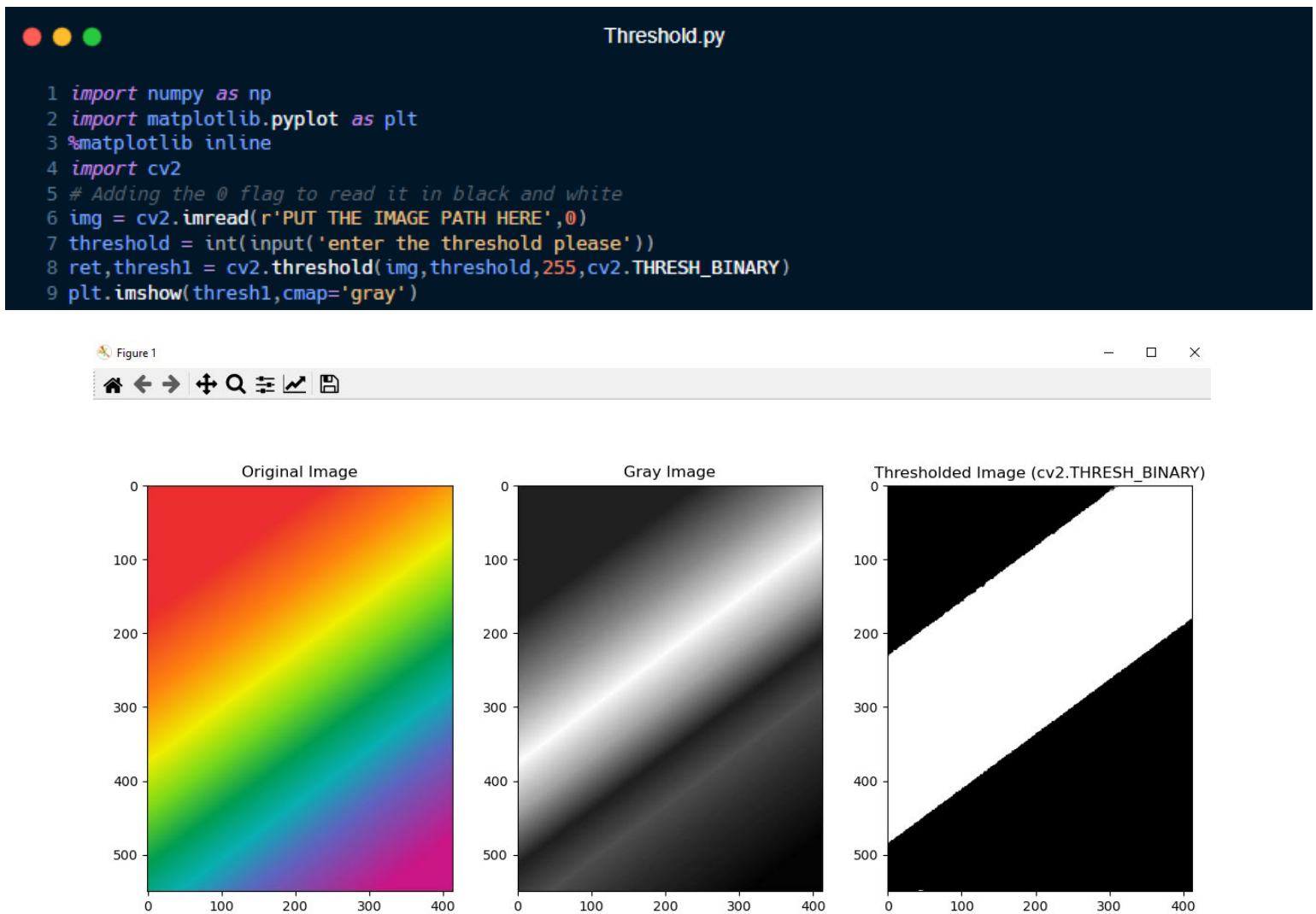


Figure 13 - Thresholding function

1.7 Histogram:

This code imports the necessary libraries numpy, matplotlib, cv2, and seaborn. It also reads an image file using cv2.imread() function, and stores it in a variable named "img".

First, the user is prompted to select an image file using the QFileDialog widget. Then, the function loads the image and converts it to RGB and grayscale using the OpenCV library.

The function opens a dialog window to select between two techniques: "Calculate Histogram" and "Histogram Equalization". If the user selects "Calculate Histogram", the function creates a subplot of four images. The first image is the original RGB image, the second image is the grayscale image, and the third image is the histogram of the RGB image. The histogram is created using the seaborn library, and a separate kernel density estimation (KDE) plot is generated for each color channel (red, green, and blue). The fourth image is the histogram of the grayscale image, which is also generated using seaborn.

If the user selects "Histogram Equalization", a second dialog window is opened to choose between "Gray" and "RGB" techniques. If the user selects "Gray", the function creates a subplot of four images. The first image is the grayscale image, the second image is the histogram of the grayscale image, the third image is the grayscale image after histogram equalization using the cv2.equalizeHist function from OpenCV, and the fourth image is the histogram of the grayscale image after histogram equalization. If the user selects "RGB", the function first converts the RGB image to HSV color space, equalizes the V channel using cv2.equalizeHist, and then converts the image back to RGB. The function creates a subplot of four images. The first image is the original RGB image, the second image is the histogram of the RGB image, the third image is the RGB image after histogram equalization, and the fourth image is the histogram of the RGB image after histogram equalization. The histograms in both cases are generated using seaborn.

Finally, if the user presses "Cancel" in the dialog windows, the function simply returns without performing any action.

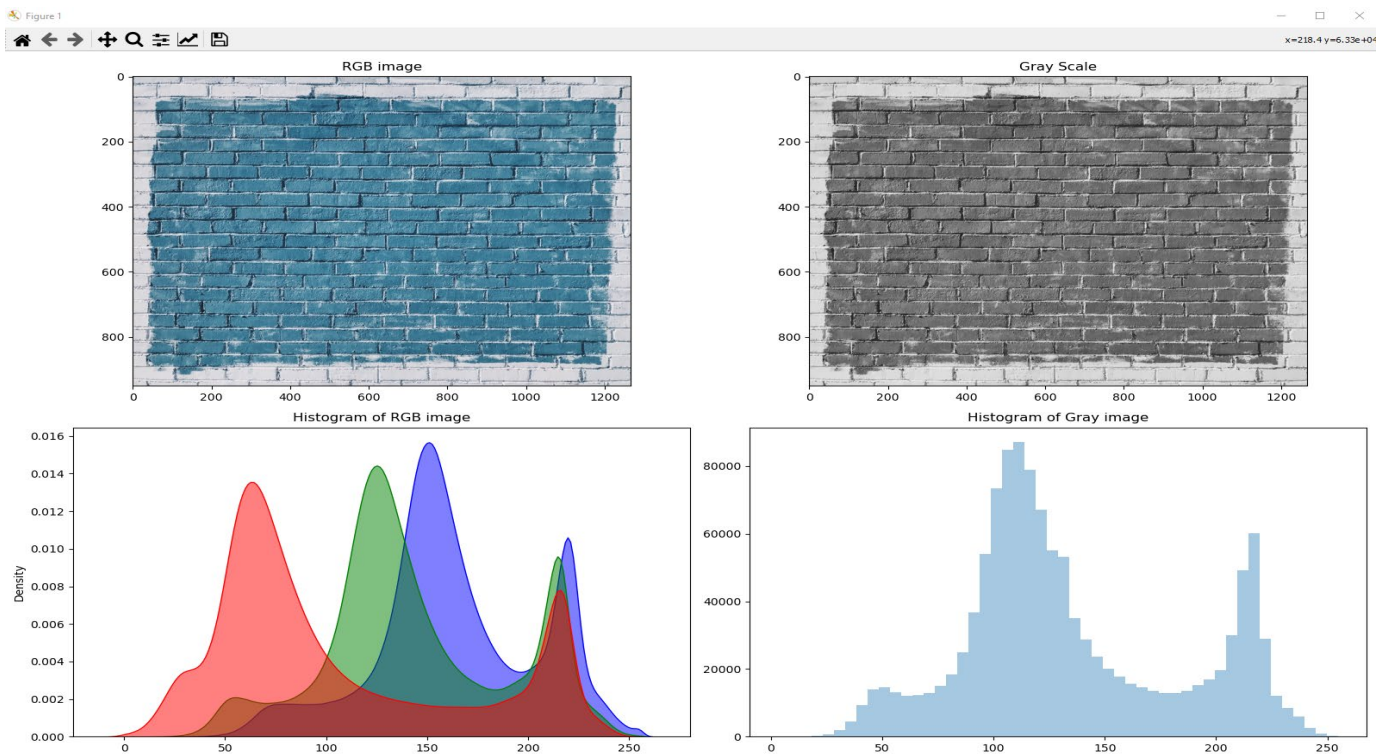


Figure 14 - Histogram function

1.8 Edge detection:

This is a Python function that performs gradient-based edge detection on an image using various techniques such as Sobel, Laplacian, and Canny edge detection. The function allows the user to select the desired technique through a dialog window and displays the result using matplotlib.

The function starts by opening a file dialog to allow the user to select an image file. Once an image is selected, the function converts the image to grayscale and then opens a dialog window to allow the user to select the desired gradient technique.

If the user selects the "Sobel x" or "Sobel y" techniques, the function applies the corresponding Sobel operator to the image and displays the resulting edge map using matplotlib.

If the user selects the "Sobel x + Sobel y" technique, the function applies both Sobel operators to the image and displays the resulting x-edge map, y-edge map, and the combined edge map using matplotlib.

If the user selects the "Laplacian" technique, the function applies the Laplacian operator to the image and displays the resulting edge map using matplotlib.

If the user selects the "Canny edge detector" technique, the function applies the Canny edge detection algorithm to the image by first blurring the image, computing the median value of the pixels, and then applying the Canny edge detector with appropriate thresholds. The function then displays the original image, the blurred image, and the resulting edge map using matplotlib.

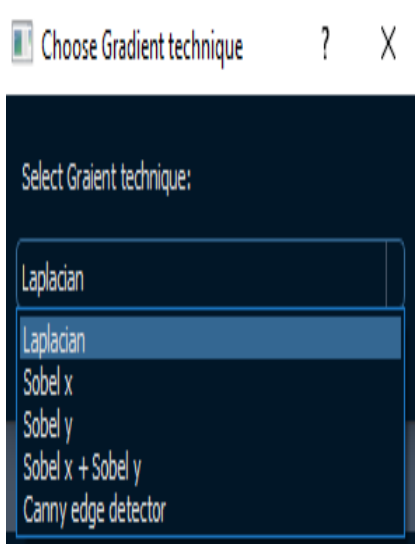


Figure 15 - Select gradient technique

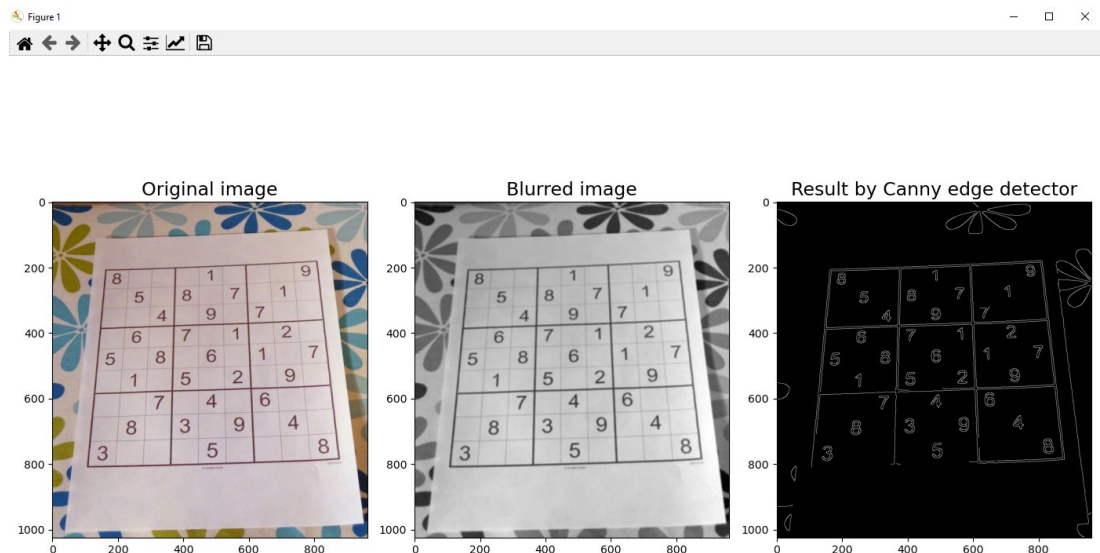


Figure 16 - Result by Canny edge detector

1.9 Feature matching:

In this project, the technique used for object detection is Feature Matching rather than Template Matching. Feature Matching is a more advanced method that can detect matching objects in an image even if the target image is not presented in the same way as the search image. This technique involves extracting key features from the input image using concepts from corner, edge, and contour detection. A distance calculation is then performed to find all the matches in a secondary image, eliminating the need for an exact copy of the target image.

Feature Matching can be performed using three methods, namely Brute-Force Matching with ORB Descriptors, Brute-Force Matching with SIFT Descriptors and Ratio Test, and FLANN-based Matcher. The most effective method among these is the combination of SIFT and FLANN-based Matcher. FLANN-based Matcher is an image-matching algorithm that performs fast nearest neighbor searches in high-dimensional spaces. It achieves this by projecting high-dimensional features to a lower-dimensional space and generating compact binary codes. This approach allows for quick image search using binary pattern matching or Hamming distance measurement, which substantially reduces the computational cost and improves search efficiency.

SIFT, which stands for Scale-Invariant Feature Transform, is a patented algorithm first introduced by D. Lowe from the University of British Columbia in 2004. SIFT is invariant to image scale and rotation, and it helps locate local features in an image, also known as key points. These key points are scale and rotation invariants that can be used in various computer vision applications such as object detection, image matching, and scene detection. SIFT features can also be used during model training as they are not affected by the image's size or orientation, which is a significant advantage over edge or hog features.

This function performs feature matching using the SIFT (Scale-Invariant Feature Transform) algorithm between two grayscale images `img1` and `img2`.

The SIFT algorithm detects and extracts features from an image, which are then represented by keypoints and descriptors. Keypoints are specific points or regions in an image that have unique features, such as edges or corners, and descriptors are numerical representations of the keypoints' features.

The code first reads in the two images using `cv2.imread` and converts them to grayscale using `cv2.cvtColor`. Then it initializes a SIFT detector using `cv2.xfeatures2d.SIFT_create()` and uses it to detect and compute the keypoints and descriptors of both images using `sift.detectAndCompute`.

After this, it sets up FLANN (Fast Library for Approximate Nearest Neighbors) parameters, which are used to perform the actual matching. It then applies the ratio test to the matches, which filters out poor matches based on the distance between the two closest matches for each keypoint.

Finally, it draws the resulting matches between the two images using `cv2.drawMatchesKnn` and saves the output to `flann_matches`.



Figure 17 - Feature matching function using sift algorithm

A Guide to Using a Desktop Application:

Certainly! To utilize the digital image processing desktop application, please follow the steps outlined below:

- i. Clone the application repository by executing the command **"git clone <https://github.com/Hossamster/Edit-Image-App.git>"** in your preferred command-line interface. This command will create a copy of the repository on your local machine.
- ii. Navigate to the cloned directory by executing **"cd Edit-Image-App"** in your command-line interface. This command will change your working directory to the cloned application folder.
- iii. Install the required packages by executing the command **"pip install -r requirements.txt"** in your command-line interface. This command will install all the necessary dependencies, including Python 3.x, PyQt5, numpy, matplotlib, seaborn, and OpenCV.
- iv. Run the application by executing the command **"python main.py"** in your command-line interface. This command will start the application.
- v. Click on the "Open Image" button within the application to select an image to work with. The application supports common image formats such as .jpg, .png, .bmp, and others.
- vi. You can then apply various image processing techniques to the selected image using the different buttons available on the application. The supported image processing techniques include resizing, grayscale conversion, inversion, thresholding, blurring, gradient calculation, histogram calculation, drawing shapes like circles or rectangles, morphological operations, blending images, and more.

By following these steps, you can utilize the digital image processing desktop application to manipulate and process images with ease.

Conclusion:

In summary, the digital image processing desktop application presents an extensive range of features that allow users to manipulate images in various ways. The application was developed using several powerful and flexible tools such as Python 3.x, PyQt5, numpy, matplotlib, seaborn, and OpenCV, which adds to its versatility and effectiveness in image processing. Users can effortlessly open an image, change its size, convert it to grayscale, invert it, apply thresholding and blurring, use gradient effects, calculate histograms, and draw shapes like circles or rectangles. The application also offers morphological operations and blending of images. To utilize the application, the user needs to clone the repository, download the required packages, and launch the application. After selecting an image, the user can proceed to apply different image processing techniques available to edit the image according to their preferences. All things considered, this application presents an excellent tool for individuals who require image manipulation for personal or professional use.