Cairo University

Faculty of Engineering

Systems and Biomedical Department

Spring 2024

Computer Vision(SBE 3230)

Assignment 1
"Filtering and Edge Detection"

Abdulrahman Emad

Mariam Ahmed Saied

Mourad Magdy

Youssef Ashraf

Ziad ElMeligy

Under the supervision of

Prof/Ahmed Badawy

Eng/Layla Abbas

Eng/Omar Hesham

Table of Contents

Table of Contents **2**
About the Project **3**
Features **3**
1. Noise 3
    Functions 3
    Experiments 4
    Observations 4
2. Filters 5
    Functions 5
    Experiments 6
    Observations 6
3. Edge Detection 8
    Functions 8
    Experiments 9
    Observations 9
4. Histograms and Distribution Curves 12
    Functions 12
    Experiments 12
    Observations 12
5. Image Equalization and Normalization 13
    Functions 13
    Experiments 13
    Observations 13
6. Local and Global Thresholding 14
    Functions 14
    Experiments 14
    Observations 14
7. Transformation of Colored Image to GrayScale 15
    Functions 15
    Experiments 15
    Observations 15
8. Frequency Domain Filters (Low pass and High pass) 15
    Functions 15
    Experiments 15
    Observations 15
9. Hybrid Image 16
    Function 16
    Experiments 16
    Observations 16

About the Project

This Task uses the Model-View-Controller (MVC) architecture pattern, the reason beyond this choice was providing a structured approach to managing the application's data, user interface and control logic. The framework used is Qt framework using C++ language and OpenCV library. This report gives insight into the functionality and the features of the application, highlighting the findings and observations while working.

Note: the application turns any uploaded image to grayscale by default when using the program methods for the ease of computation.

About MVC architecture
The Model-View-Controller (MVC) architectural pattern divides an application into three interconnected components:

1. **Model**: which represents the application's data and logic
2. **View**: responsible for presenting the data to the user and handling user interactions.
3. **Controller**: which processes user input, updates the model, and manages the communication between the model and the view.

By separating concerns, MVC promotes modularity and maintainability, allowing for easier development, testing, and modification of each component independently. This structure facilitates scalability, code reuse, and a clear separation of presentation and business logic in software development.

Features

1. Noise

Functions

1- Salt and Pepper Noise
The method for adding Salt and Pepper noise is straightforward. It relies on a probabilistic approach to decide whether a pixel should be assigned a specific value or not. For each pixel in the image, a random number is generated. If this random number equals 1, the pixel is set to the maximum intensity value(white), representing "salt." If the random number equals 2, the pixel is set to the minimum intensity value(black), representing "pepper." This process is repeated for all pixels in the image, resulting in the addition of random white and black speckles, mimicking the appearance of salt and pepper scattered throughout the image.

2- Uniform Noise
The approach for adding Uniform noise involves generating random values uniformly distributed within a specified range and adding them to the original image.
In this function, a copy of the input image img is created to store the result. Then a new matrix noise with the same size and data type as the input image is initialized to hold the uniform noise.
The 'randu()' function provided by OpenCV is used to generate random values uniformly distributed between 0 and 50. These random values are stored in the noise matrix.
Next, the generated uniform noise matrix noise is added to the duplicate image result. This addition operation is performed element-wise, meaning each pixel in the noise matrix is added to the corresponding pixel in the duplicate image.

Finally, the resulting image result, now containing the added uniform noise, is returned. This process introduces random variations throughout the image, simulating the appearance of uniform noise.

3- Gaussian noise

Adding Gaussian noise involves generating random values from a Gaussian distribution with specified mean and standard deviation parameters and adding them to the original image.
A new matrix noise with the same size and data type as the input image is initialized to hold the Gaussian noise.'The randn()' function provided is utilized to generate random values from a Gaussian distribution with the specified mean and sigma. These random values are stored in the noise matrix.
After generating the noise, it is added to the image. This addition operation is performed element-wise, meaning each pixel in the noise matrix is added to the corresponding pixel in the duplicate image.
Finally, the resulting image result, now containing the added Gaussian noise, is returned. This process introduces random variations throughout the image, simulating the appearance of Gaussian noise.

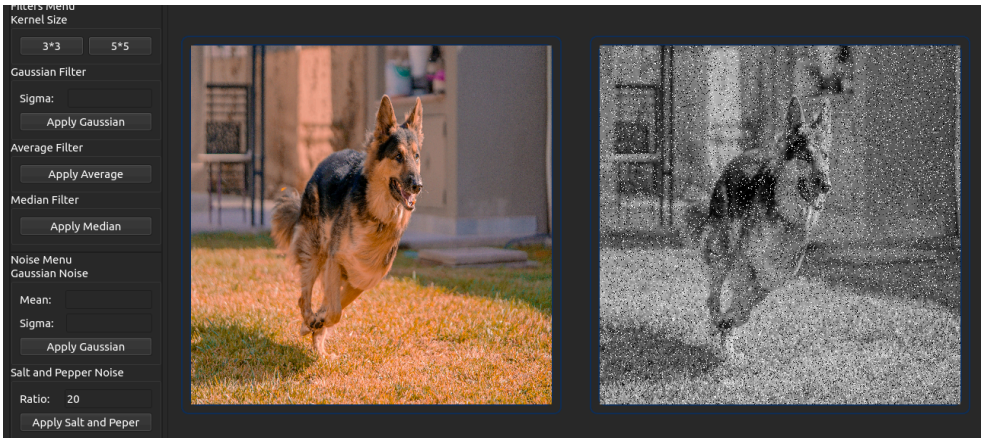
Experiments

One problem we faced when implementing the salt and pepper filter was because of the difference between the implementation of gray scaled images and colored images as we customized the functions to work with grayscale images but the input was three channeled images which caused the problem to crash, when debugging and analyzing this inconsistency was found.

Observations

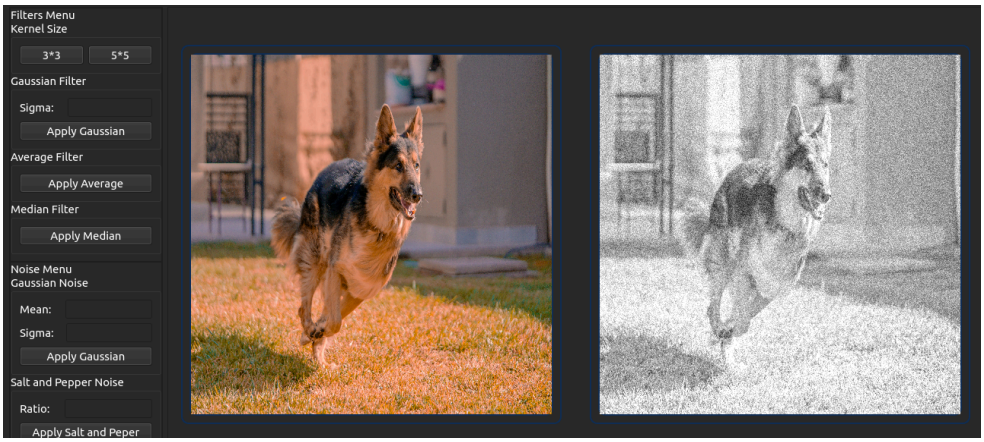
Salt and Pepper Noise

Increasing noise density leads to more pronounced salt and pepper artifacts, resulting in image degradation and reduced visual quality. And Noise removal techniques such as median filtering are effective in mitigating salt and pepper noise while preserving image details.



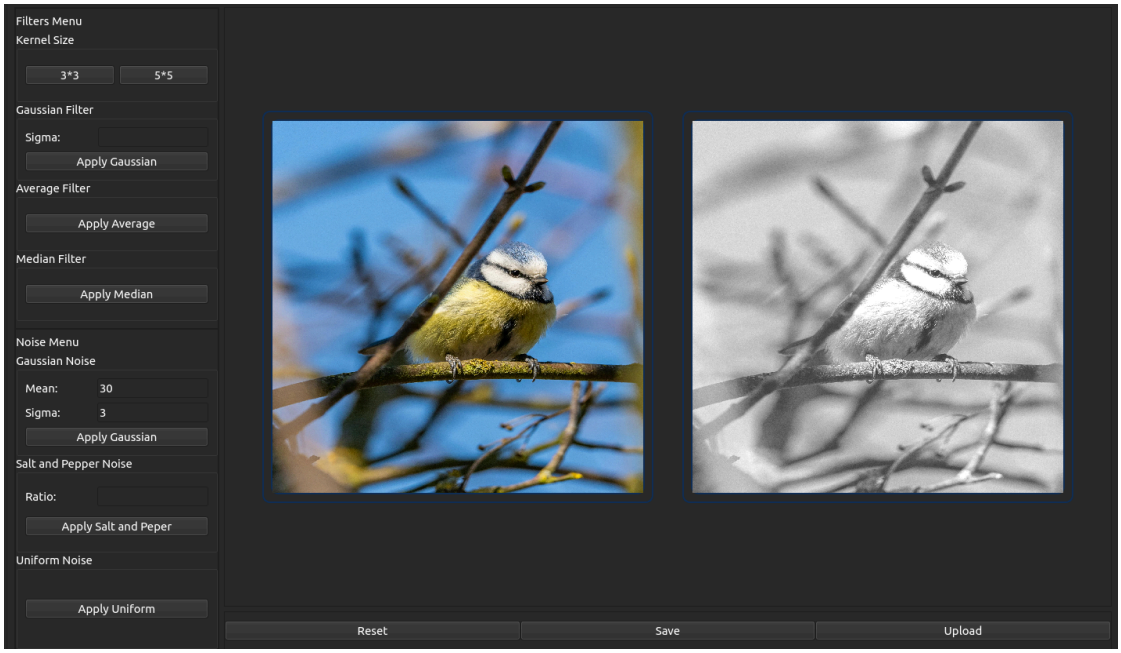
Uniform Noise

Higher intensity levels of uniform noise introduce more noticeable artifacts and degrade image clarity.
Changes in distribution parameters alter the uniform noise pattern and affect the overall texture and appearance of the image.



Gaussian Noise

Higher variance values result in greater image distortion and decreased perceptual quality due to the increased spread of noise.



2. Filters

Functions

1- Median Filter:

The median filter is a spatial domain filter commonly used for noise reduction in images. It replaces each pixel's value with the median value of the pixels in its neighborhood.
In this function, a copy of the input image is stored for the operations.
The **kernelSize** parameter determines the size of the neighborhood around each pixel.
For each pixel in the image, the function extracts a square region centered around that pixel, with a size defined by the kernelSize. This region is referred to as the kernel.The kernel is then reshaped into a column vector to facilitate sorting. The nth element algorithm is used to efficiently find the median value within the kernel.
Once the median value is determined, it is assigned as the new value for the corresponding pixel in the resulting image.This process is repeated for every pixel in the image, excluding the border pixels where the kernel may extend beyond the image boundaries.

2- Gaussian Filter:

The Gaussian filter is a popular image smoothing technique that blurs an image by convolving it with a Gaussian kernel. This operation reduces noise and removes high-frequency components.
A Gaussian kernel is generated using the gaussianKernel() function, which produces a 2D matrix representing a Gaussian distribution with a specified kernel size and standard deviation .Then we convolved the input image with the generated Gaussian kernel. This function performs a 2D spatial filter operation on the image, effectively applying the Gaussian blur.

The parameters -1 and Point(-1, -1) indicate that the output image will have the same depth as the input image and that the center of the kernel will be considered as the anchor point for convolution, respectively. 0 specifies that no additional scaling will be applied to the kernel.
The BORDER_DEFAULT parameter specifies the border mode used for extrapolation when the kernel goes outside the image boundaries during convolution.

3- Average Filter:
The average filter, also known as the box filter, is a simple image smoothing technique that replaces each pixel's value with the average value of its neighboring pixels.
A kernel matrix is initialized using the Mat::ones() function, creating a kernel filled with ones of size kernelSize x kernelSize. Each element in the kernel matrix represents the contribution of a neighboring pixel to the average.
The kernel matrix is then divided by the total number of elements in the kernel to normalize the filter. This normalization ensures that the resulting pixel values remain within a reasonable range.
Then we convolve the input image with the generated kernel.
This function performs a 2D spatial filter operation on the image, effectively applying the average filter.
Finally, the resulting image with the average filtering applied is returned.

Experiments

The median filter needed optimization that we did was to use matrices instead of loops when implementing the median filter as it’s a nonlinear method so it’s computationally intensive and needed optimization

Observations

Median Filter
Increasing the kernel size generally leads to better noise reduction but may result in loss of image details, especially in regions with fine structures.
The median filter effectively preserves edges and sharp transitions in the image while smoothing out noise artifacts.
Computational cost increases significantly with larger kernel sizes due to the need for sorting operations within the kernel.

Gaussian Filter
Higher sigma values result in stronger blurring and smoother image appearance, effectively reducing noise but potentially causing loss of fine details.
Gaussian filtering offers a good balance between noise reduction and preservation of image features, particularly with moderate sigma values.
Larger kernel sizes lead to more extensive blurring and smoother transitions but also increase computational complexity.

Average Filter
The average filter provides efficient noise reduction while maintaining image sharpness, especially with smaller kernel sizes.
Increasing the kernel size improves noise reduction but may result in noticeable blurring of image details.
Computational efficiency remains relatively stable across different kernel sizes, making the average filter suitable for real-time applications.

We also tried filtering this exact image with each filter and compare their outputs

The image to be filtered (added salt and pepper noise)

Noise removal using Median filter

Noise removal using Average Filter

Noise removal using Gaussian filter

Observation

The one which appeared to give the best performance when it comes to noise removal in case of salt and pepper is the Median filter, both the gaussian and the average filter have very close results as they are both have very similar kernels so they give similar results, the median filter is the best one of them is median because of its robustness to extreme outliers and salt and pepper noise are basically extreme outliers

3. Edge Detection

Functions

Different edge detection functions are implemented within the ‘EdgeDetection’ class

1- detectEdgeSobel Function:
Detects edges in the input image using the Sobel operator.
Calls separate functions ‘detectEdgeSobelX’ and ‘detectEdgeSobelY’ to compute gradients along the X and Y directions.
Calculates the magnitude of gradients and converts it to a 8-bit unsigned integer (CV_8U) for visualization.

2- detectEdgeRobert Function:
Detects edges in the input image using the Robert operator.
Calls separate functions ‘detectEdgeRobertX’ and ‘detectEdgeRobertY’ to compute gradients along the X and Y directions.
Calculates the magnitude of gradients and converts it to a 8-bit unsigned integer (CV_8U) for visualization.

3- detectEdgePrewitt Function:
Detects edges in the input image using the Prewitt operator.
Calls separate functions ‘detectEdgePrewittX’ and ‘detectEdgePrewittY’ to compute gradients along the X and Y directions.
Calculates the magnitude of gradients and converts it to a 8-bit unsigned integer (CV_8U) for visualization.

4- detectEdgeCanny Function:
Detects edges in the input image using the Canny edge detection algorithm. Applies Canny edge detection with specified lower and upper thresholds to obtain the edge-detected image.

5- edgeMagnitude Function:
Computes the edge magnitude from gradients along the X and Y directions.
Takes gradient images along X and Y directions as input and computes the magnitude using the Euclidean distance formula.

Experiments

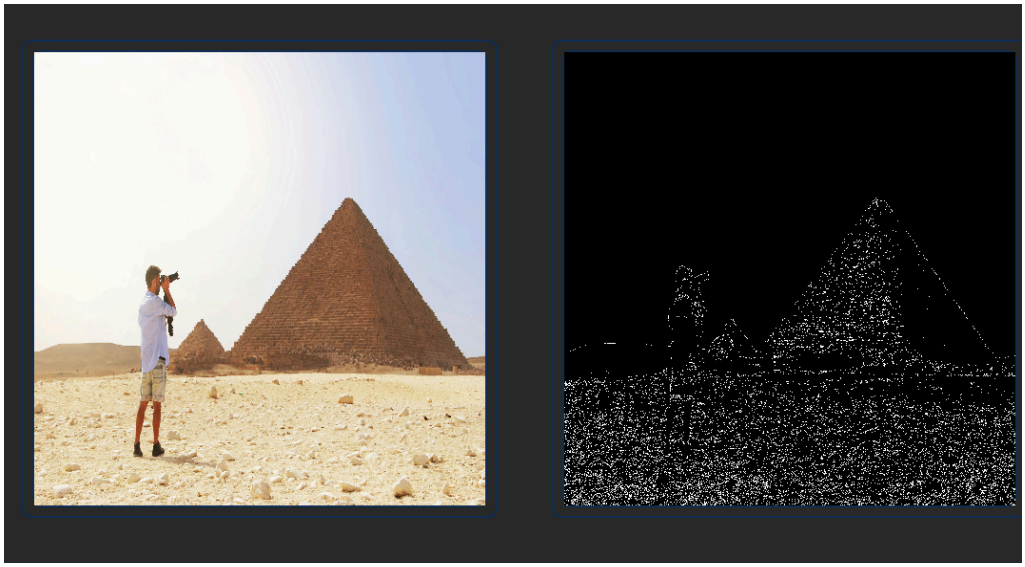
We tried Testing with a variety of input images to evaluate the effectiveness of different edge detection operators in detecting edges under various conditions.
We also compare edge-detected images generated by different operators to assess their performance in capturing edges accurately.

Observations

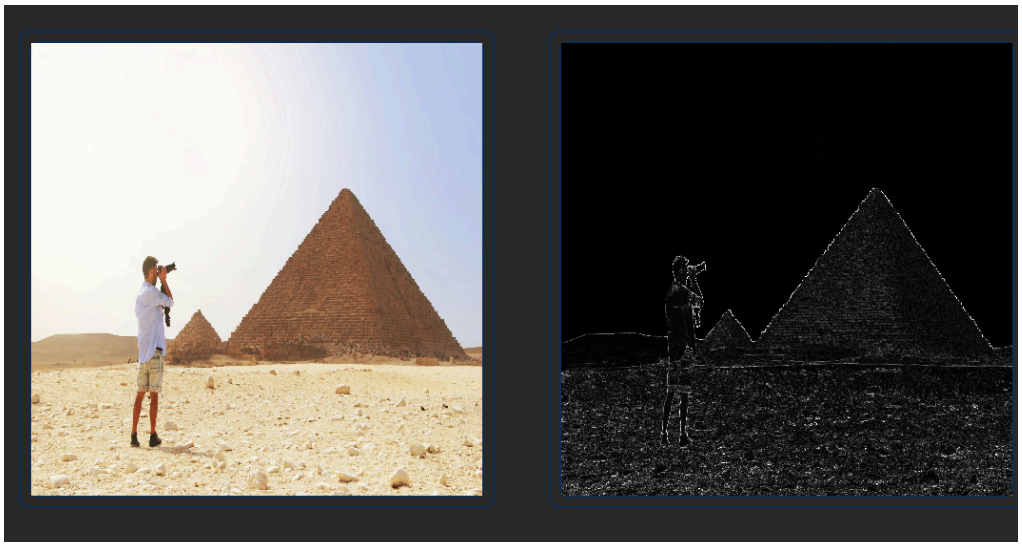
Operator Performance:
Different edge detection operators exhibit varying performance in detecting edges, with some being more sensitive to noise or more suitable for specific edge characteristics.
Accuracy:
Sobel and Prewitt operators typically provide good edge detection results for general-purpose applications, while the Canny edge detector offers better noise suppression and localization of edges.
Computational Complexity:
The computational complexity of each method varies, with some operators being more computationally intensive than others. For real-time applications, consideration of processing time is essential.
Parameter Sensitivity:
Some edge detection methods, such as the Canny edge detector, require careful selection of threshold parameters to achieve optimal results, which may vary depending on the image content and noise level.
Seeing the different results of different edge detectors:
Robert edge detection

Prewitt edge detection

Sobel edge detection



Canny edge detection



Comparing different edge detection techniques

Sobel	Canny	Prewitt	Robert
Detects edges by emphasizing regions with high spatial intensity gradients.	Highly effective edge detection method. Involves multiple stages including noise reduction, gradient calculation, non-maximum suppression, and edge tracking. Relatively	Similar to Sobel but uses a slightly different kernel for gradient calculation.	Simple and computationally efficient.
Computationally efficient with 3x3 convolution kernels.	Produces well-defined edges with adjustable width.	Computationally efficient with 3x3 convolution kernels.	Uses small 2x2 convolution kernels.
Sensitive to noise and variations in illumination.	computationally intensive but robust to noise and illumination variations.	Sensitive to noise and variations in illumination.	Highly sensitive to noise and variations in illumination, may produce thinner edges.

Thus, while Sobel, Prewitt, and Roberts offer simplicity and efficiency, Canny stands out for its robustness and accuracy, particularly in scenarios with noise and varying illumination.

4. Histograms and Distribution Curves

Functions

It's implemented in The 'Histogram' class

distributionCurve Function:
This function plots the distribution curve based on the histogram data provided.
It constructs a curve image with the specified color channels (b, g, r) and draws the distribution curve based on the normalized histogram data.

rgbHistogram Function:
This function calculates the RGB histograms of the input image.
It separates the color channels, calculates histograms for each channel, normalizes them, and plots them to generate the RGB histogram visualization.

Experiments

There are other methods and implementations of the cumulative distribution function (CDF) and adjusting pixel values, like contrast stretching, which linearly stretches the histogram to the full intensity range. Another

approach is adaptive histogram equalization, which divides the image into smaller regions and performs histogram equalization locally, allowing for better preservation of local contrast.
However, The method we used is widely used for its simplicity, efficiency, and effectiveness in improving image quality, making it a suitable choice for general-purpose histogram equalization tasks.

Observations

Distribution Curve Plotting: The ‘distributionCurve’ function accurately plots the distribution curve based on histogram data, providing visual insights into the pixel intensity distribution.
RGB Histogram Visualization: The ‘rgbHistogram’ function accurately calculates and plots RGB histograms, facilitating the analysis of color distribution in images.
Impact on Image Quality: Histogram equalization significantly improves the visual quality of images with low contrast or uneven lighting conditions, enhancing details and improving overall appearance.

5. Image Equalization and Normalization

Functions

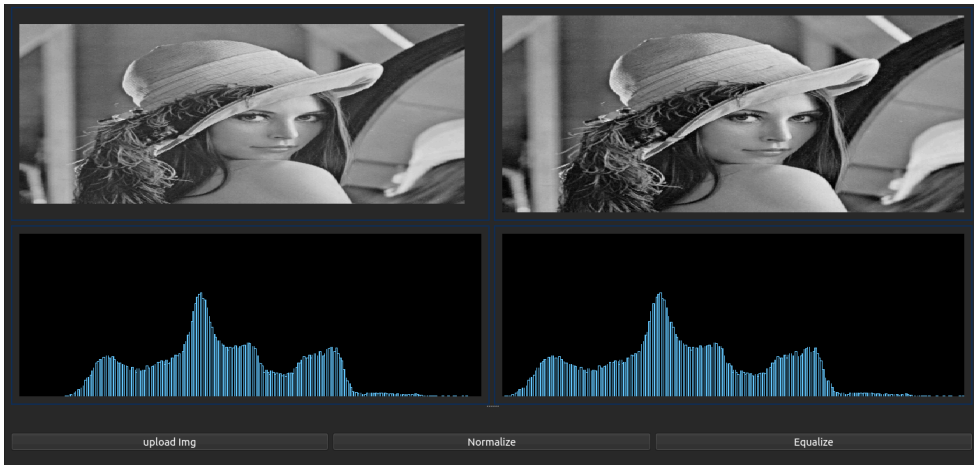
Image normalization and equalization are implemented within the histogram class

Normalize image:
This function normalizes the input image, scaling its pixel values to fit within the range [0, 255] using OpenCV function ‘normalize’ with parameters `NORM_MINMAX` and data type ‘CV_8U’ for conversion.

Equalize Image Function
This function performs histogram equalization on the input grayscale image, It first converts the input image to grayscale if it's in color, Histogram equalization is applied by calculating the cumulative distribution function (CDF) of the image's histogram.
The CDF is then normalized and used to adjust the pixel values in the input image, resulting in an equalized image.

Experiments

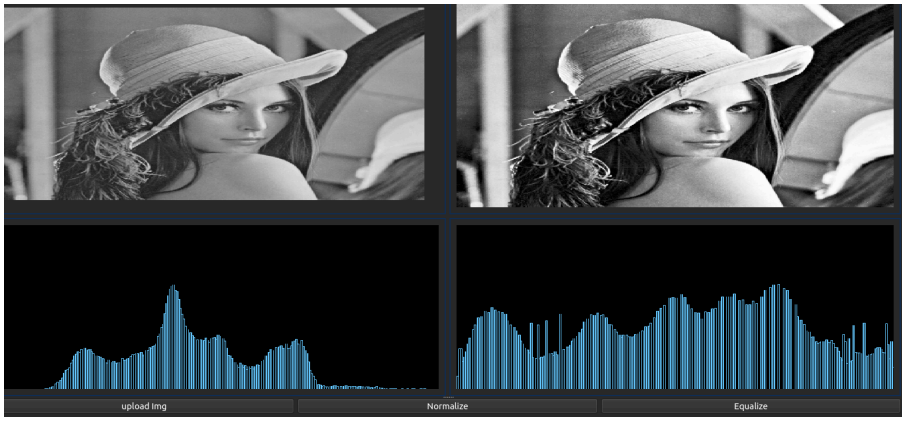
We tried testing the OpenCV function using various input images with different lighting conditions and contrasts to see the effectiveness of normalization and histogram equalization.
Evaluation of the impact of histogram equalization on enhancing image contrast and improving visual quality.
Comparison of normalized and equalized images with their original counterparts to measure the effectiveness of the processing.



Observations

Image Normalization:
We found that the normalization of the image doesn’t really change the histogram and the image changes only slightly, what we noticed is a very slight change in the grayscale levels of the image,(it just gets a little darker)

Image equalization:
it changes the distribution of the gray levels of the image and changes the histogram, the distribution of the gray levels spreads out and the vast contrast between the extremely dark and the extremely light shrinks making the picture more uniform.



6. Local and Global Thresholding

Functions

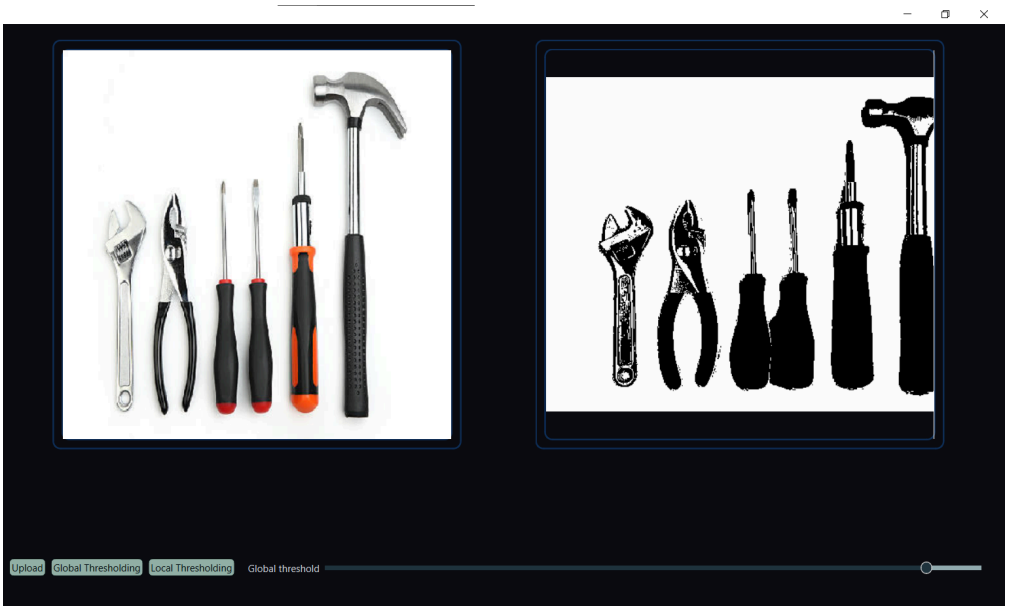
Global thresholding:
Global thresholding sets a single threshold value for the entire image. Pixels with intensity values higher than this threshold are set to a predefined maximum value (in this case, 250), while those below the threshold are set to zero.
The function iterates through each pixel of the image, comparing its intensity value with the given threshold. If the intensity value is greater than or equal to the threshold, the pixel is set to the maximum value (250), otherwise, it is set to zero.

Experiments

There were other implementation options for thresholding like Otsu's Thresholding, Adaptive Thresholding, and Histogram-based Thresholding but we chose this one due to its simplicity and ease of understanding

Observations

We observed that Global thresholding converts a grayscale image into a binary image with high contrast. The global thresholding creates a black-and-white image where objects stand out distinctly against the background, which makes edge detection and analysis of specific features within the image much easier.



7. Transformation of Colored Image to GrayScale

Functions

The function loops over the matrix of the image and calculates the grayscale value of each pixel from the colored image using the weighted sum method where the weights for each color channel are provided by the formula:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

Experiments

There are other possible implementations other than weighted average like simple averaging, lightness method, single channel extraction, but we chose weighted average because it is simple yet effective and it strikes a balance between computational efficiency and perceptual accuracy which will be handy as images get more detailed and larger.

Observations

The function was successful in turning any colored image to grayscale with quick running time

8. Frequency Domain Filters (Low pass and High pass)

Functions

The function applies frequency domain filtering to an image using discrete fourier transform, first you convert the image to 32-bit floating point format to ease the DFT operation, then compute the DFT, and then generate a mask based on the radius given, to specify the region, then apply the filter based on the type (low pass filter or high pass filter) as for low pass filter the frequencies outside of the mask and the high pass filter inside of the mask

Experiments

The implementation method was pretty straight forward the problems we faces was with the different data types as some of them caused problems in the code
Dealing with the output of the DFT lead to further experimentation with the compatibility of the output of the DFT and the matrix itself of the image and the QPixmap

Observations

The function effectively suppresses high frequencies beyond the specified radius in low-pass filtering.High-pass filtering successfully enhances edges and high-frequency details within the circular mask.The choice of filter radius significantly influences the degree of filtering and the resulting image quality.Filtering artifacts may occur, especially near sharp transitions or edges, depending on the filter type and radius.

9. Hybrid Image

Function

The function used for Mixed images to output a hybrid image uses a fourier transform to apply high pass filter to one image and low pass filter to the other, then performs element wise addition to mix the high pass elements of the image with the low pass elements of the other.

Note: The high pass and low pass filters are done using the already implemented frequency domain filters

Experiments

There were a bunch of options when we were implementing the hybrid function

1. To use element wise addition without filtering
2. To use sobel for high pass and the gaussian for a low pass filter
3. To use fourier transform for frequency filtering.

The first option just overlaid the two images on top of each other, which wasn't the best option because it wasn't really a hybrid image, it was just two overlaid images, so that created the need for processing the images using frequency filters first then adding them element wise.
The second option gave an output of a hybrid image but the quality wasn't the best, one image that was smoothed by the gaussian was very vivid, and the other had very weak edges, the problem was kind of solved when trying to add thresholding and tweaking the values.
The third option was similar to the second but it gave much better results, as it needed much less parameters and tweaking the hybrid image was closer to the standard known output than the second option

Observations

What we observed from the output of the hybrid image is that we can see both images at the same time, if you squint your eyes you can see the lowpass picture more and it if you focused more on the image you'll see the highpass image