Image Analysis:- Implementing filters with Feature Detection and Matching

1st Hussain Kanchwala
Mechanical and Industrial Engineering Department
Northeastern University
Boston, USA
kanchwala.h@northeastern.edu

Special Thanks
I would like to extend my special thanks to Prof. Bruce A. Maxwell for his invaluable guidance for the successful completion of this project.

***Abstract*—This project encompasses a range of image processing techniques, from fundamental image filtering to more complex applications like face detection. A significant aspect of the work is the development of the Harris Corner Feature Detector. This implementation underlines the essential principles of image feature detection and lays the groundwork for exploring feature matching concepts.**

An important extension of this project is the use of pixel intensity in the neighborhood as a descriptor for feature matching. This approach harnesses the local intensity patterns around key points, identified by the Harris Corner Feature Detector. By examining pixel intensity variations in these regions, the algorithm achieves nuanced and accurate matching across different images.

This project, through its development and integration of these methods, delves into the complexities of image analysis and pattern recognition, offering valuable insights for practical applications in the field.

Please visit my GitHub Repository for more information and source code.

CONTENTS

I. INTRODUCTION

This project initiates with an insightful introduction to the OpenCV library, focusing on its capabilities in video and image capture. As a foundational tool in computer vision, OpenCV's role is pivotal in facilitating the exploration and application of various image processing techniques. The core of the project lies in the implementation of 14 distinct image filters, covering a spectrum from basic grey scaling to more complex methods such as Sobel filtering and Gaussian smoothing. Furthermore, the project delves into the realms of face detection and then into feature detection and matching, crucial components in the field of visual Simultaneous Localization and Mapping (SLAM). This comprehensive study not only demonstrates the practical utility of these filters in image processing but also highlights their significance in advancing SLAM technologies, an essential aspect of modern computer vision.

II. EASE OF USE

To begin using the tools provided in this project, first clone the GitHub repository and build the package. Launch the application by typing ‘./vidDisplay‘ in your terminal. This command activates your camera, allowing you to apply and visualize various filters in real-time on your camera feed. Follow the on-screen key press instructions for navigating through different filters.

***NOTE:-** Exercise caution when pressing the key ’e’. This key activates the custom implementation of the Harris Corner Feature Detector, which is computationally intensive. Due to its high processing demand, it is not ideally suited for real-time operations and should only be used with a powerful processor.

For feature matching, run ‘featurematching.cpp‘ by typing ‘./matching‘ in the terminal. You will need to provide paths for the reference and target images. The program will then proceed to detect features on these images and match them accordingly.

A. Maintaining the Integrity

When utilizing code from this repository, it is mandatory to acknowledge and provide credit to Prof. Bruce A. Maxwell and Hussain Kanchwala. Proper attribution is essential for maintaining the integrity of the work and respecting the contributions of its creators.

III. Reading the Image

OpenCV, a widely used library for computer vision, provides a suite of powerful functions for image processing. Key among these are ‘imread‘ and ‘imshow‘, essential for reading and displaying images, respectively.

The ‘imread‘ function is incredibly useful for loading an image from a specified file path. Its usage is straightforward; by providing the path to an image file, the function reads the image and returns it in a format suitable for further processing.

Once an image is read, it can be displayed in a window using the ‘imshow‘ function. This function is particularly beneficial for visualizing the effects of various image processing operations.

Furthermore, OpenCV’s ‘waitKey()‘ function plays a pivotal role in interactive applications. It waits for a specified time for a user input and is commonly used to detect key presses. This feature is instrumental in enabling users to interact with the application, such as initiating specific image processing operations through the press of a key.

Together, these functions form the backbone of many image processing tasks, facilitating easy manipulation and analysis of images.

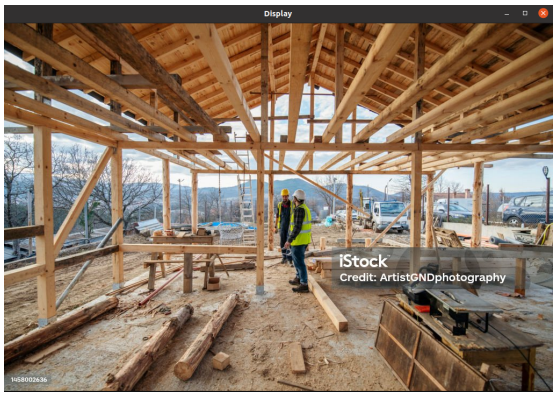

Fig. 1. Window displaying the image

IV. Filters

Filters are a fundamental component in computer vision, crucial for processing and analyzing images to extract meaningful information or enhance their characteristics. They find applications in a variety of image processing tasks, such as noise reduction, edge detection, sharpening, blurring, and more.

In the initial phase of my project, I leveraged OpenCV, a powerful library for computer vision, to open video channels and capture frames in a real-time loop. An essential aspect of this setup was the use of OpenCV’s waitKey() function. This function allowed me to create an interactive application where the user can activate different filters at the press of a key, effectively changing the processing applied to the video stream in real-time.

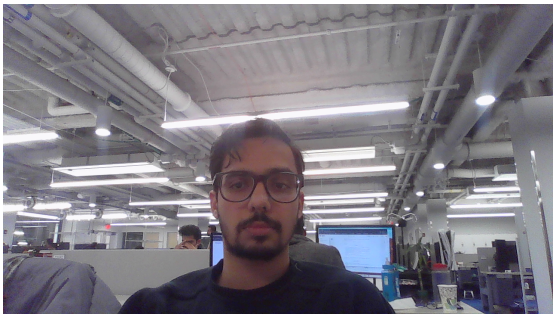

Fig. 2. Shot from Video stream

Following this setup, I implemented various common filters, each demonstrating a specific aspect of image manipulation and analysis:

A. Gray Scale and Custom Gray Scale

In this project, two variations of a grayscale filter were implemented on a live video stream using OpenCV. These implementations demonstrate different approaches to achieve a grayscale effect, each with its unique characteristics.

Standard Grayscale Filter: The first implementation utilizes OpenCV’s cvtColor function to convert the live video stream into grayscale. This conversion is a standard method in image processing, where the color image is transformed into shades of gray, reflecting the intensity of light. The grayscale filter is activated by pressing the ’g’ key when running the vidDisplay function. This approach provides a true grayscale image by averaging the color intensities or using a weighted average that reflects human perception of different colors.

Custom Blue-Inverted Grayscale Filter: The second approach involves a custom grayscale filter that uniquely manipulates the blue channel of the color image. Instead of converting the image into standard grayscale, this method subtracts 255 from the blue component of each pixel in the frame, effectively inverting the blue channel intensity. The inverted intensity values of the blue channel are then used to set all three color channels (red, green, and blue) of the destination image. The result is a grayscale-like image, but it is distinctively based on the inverted intensity of the blue channel of the source frame. This custom filter starts with the press of ’h’ key while running the vidDisplay function.

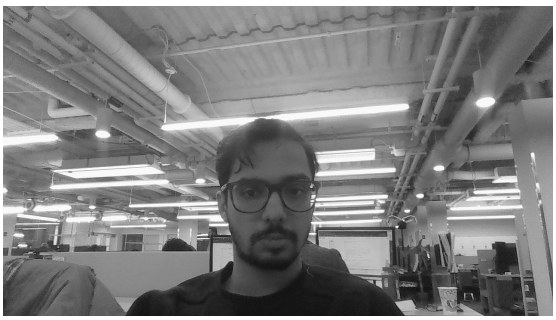

Fig. 3. Gray Scale using cvtColor

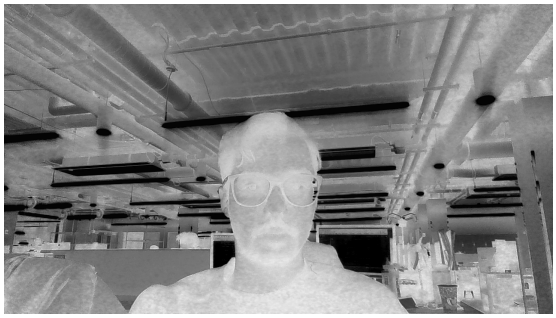

Fig. 4. Custom Gray Scale

B. Sepina Tone Filter

A sepina tone filter gives a warm, brownish tone reminiscent to a picture or a video frame in our case. It calculates a new color intensity values for the blue channel using 0.189*b+0.769*g+0.393*r where b,g and r stands for blue, green and red. For the red channel the intensity value is calculated using 0.131*b+0.534*g+0.272*r and for green channel using 0.168*b+0.686*g+0.349*r.

Fig. 5. Sepina Tone Filter

C. Gaussian Blur Filter

In this project, the Gaussian blur filter was implemented using two distinct methods: normal convolution and a separated filter approach. These methods highlight the efficiency gains that can be achieved through thoughtful algorithmic design.

Gaussian Blur via Normal Convolution: The first implementation uses the traditional convolution method with a Gaussian kernel. The kernel used is a 5x5 matrix with values [1,2,4,2,1;2,4,8,4,2;4,8,16,8,4;2,4,8,4,2;1,2,4,2,1]. This kernel is convoluted with the image to apply the Gaussian blur, a process known for its effectiveness in reducing image noise and details.

Gaussian Blur via Separated Filter: The second approach uses a separated filter method. In this method, the Gaussian kernel is split into two one-dimensional vectors [1,2,4,2,1] and its transpose. By applying these one-dimensional filters sequentially (one for the horizontal pass and one for the vertical pass), the Gaussian blur is achieved more efficiently. This separation significantly reduces computational complexity, leading to faster processing times.

The performance difference between these two methods is notable. For example, applying the Gaussian blur 10 times to the same image took 0.34 seconds using the normal convolution method, but only 0.00334 seconds using the separated filter approach. This substantial reduction in processing time demonstrates the effectiveness of the separated filter method in optimizing image processing tasks, particularly for real-time applications where speed is crucial. In order to visualize the blurring on video stream press 'b' key while running the vidDisplay.cpp.

Fig. 6. Gaussian Blur using separated filter kernel

D. Sobel Filter

The Sobel filter is a widely used technique in image processing for edge detection. In this project, it has been implemented to detect edges in a live video stream. The Sobel filter operates by emphasizing edges in specific directions, using convolution with appropriate kernels.

Vertical Edge Detection: To detect vertical edges, the Sobel filter is applied in the x-direction. This is achieved using the kernel [1,0,1;2,0,2;1,0,1]. When convoluted with the image, this kernel highlights vertical edges by taking the derivative in the x-direction. In the interactive video stream application, pressing the 'x' key activates this filter, allowing the user to visualize vertical edges in real time.

Horizontal Edge Detection: For detecting horizontal edges, the Sobel filter is applied in the y-direction. This requires the transpose of the previous kernel giving [1,2,1;0,0,0;1,2,1]. This kernel focuses on the derivative in the y-direction, effectively bringing out horizontal edges in the image. By pressing the 'y' key in the video stream application, users can switch to this mode to observe horizontal edge detection.

These implementations of the Sobel filter demonstrate its effectiveness in isolating edges in specific orientations, an essential aspect of many computer vision tasks. The ability to switch between vertical and horizontal edge detection in real-time offers an interactive way to understand how edge detection algorithms work and their impact on image analysis.

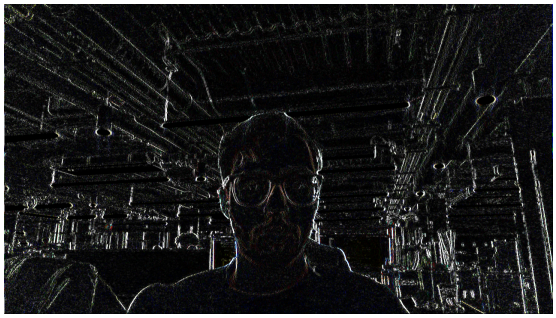

Fig. 7. Sobel Filter to detect Vertical edges

Fig. 8. Sobel Filter to detect horizontal edges

E. Gradient Magnitude

An approach to calculate and visualize the gradient magnitude of an image is implemented. This method is essential for understanding the spatial variations within the image. The gradient magnitude at each pixel is computed using the SepinaX (S_x) and SepinaY (S_y) values, which denote the vertical and horizontal components of the gradient, respectively.

The gradient magnitude is determined using the Euclidean distance formula:

$$\text{Gradient Magnitude} = \sqrt{S_x^2 + S_y^2} \tag{1}$$

This computation combines the horizontal and vertical gradient components to yield a comprehensive measure of spatial change intensity at each pixel.

To enhance the visualization and interpretation of these spatial changes, the JET colormap has been integrated. The JET colormap effectively represents the range of gradient magnitudes through a color spectrum, enabling clearer visual differentiation of variations.

An interactive feature has been incorporated into the live video stream application. By pressing the 'm' key, users can activate the gradient magnitude visualization for the current frame.

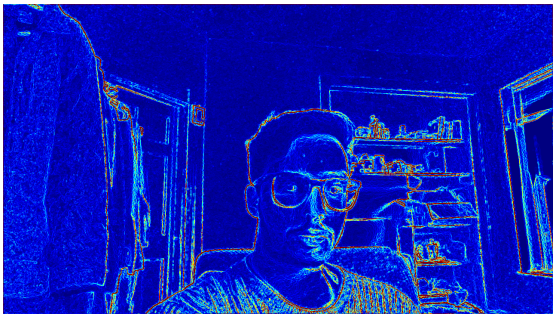

Fig. 9. Gradient Magnitude

F. Blur Quantize

The blur quantize filter applies blurring effect and then quantizes the color values across BGR into specified number of levels. This is done by dividing the color values into bins, rounding the nearest bin and then scaling back. This creates a posterization effect with reduced color levels. Press 'i' to see blur quantizing on video stream.

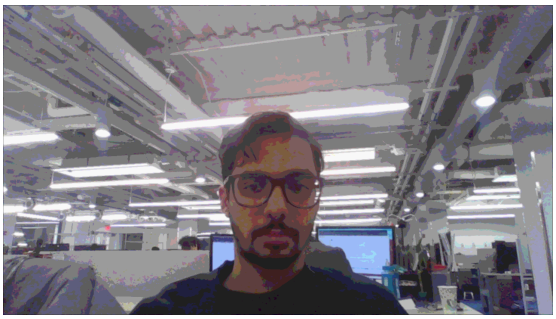

Fig. 10. Blur Quantizing Filter

G. Face detection with background gray scaling and blurring

An integral feature of this project is the implementation of real-time face detection in a video stream. This functionality is activated by pressing the 'f' key. The face detection utilizes advanced algorithms to accurately identify faces within the video frames, offering a dynamic and responsive user experience.

Building upon the face detection feature, two innovative enhancements have been introduced:

- **Selective Background Blur:** By pressing the 'a' key, users can engage a feature that blurs the background while keeping the detected face sharp and in focus. This effect emphasizes the face in the video, creating a visually appealing depth-of-field effect that is commonly seen in professional photography.
- **Selective Color Desaturation:** Pressing the 'c' key activates a unique effect where the background is converted to grayscale, while the detected face retains its original colors. This selective desaturation creates a striking contrast between the subject and the background, highlighting the face with a splash of color amidst a monochromatic backdrop.

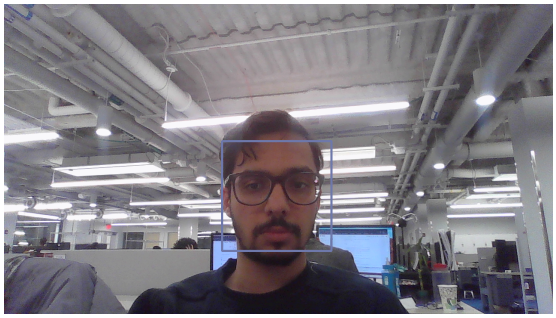

Fig. 11. Face Detection

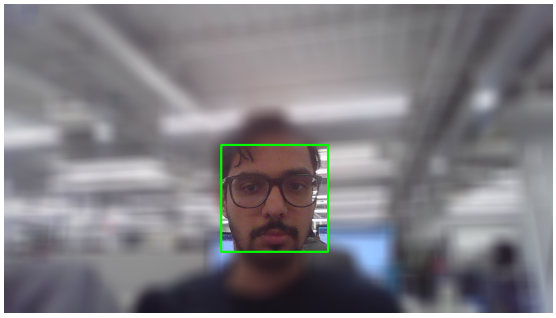

Fig. 12. Selective Background Blur

Fig. 13. Selective Color Desaturation

H. Brightening and Cartooning

1) Brightening Filter: The Brightening Filter is designed to enhance the brightness of an image uniformly across all the BGR (Blue, Green, and Red) channels. This filter incrementally increases the intensity of each channel, resulting in a brighter and more vibrant image. Users can activate this effect in the video stream by pressing the 'n' key, observing the immediate brightening impact on the live feed.

2) Cartooning Filter: The Cartooning Filter transforms the appearance of frames into a cartoon-like style. The implementation of this filter involves several key steps:

a) *Edge Preservation using Bilateral Filter:* The Bilateral Filter is employed due to its edge-preserving properties, which are essential in maintaining the defining lines of the cartoon effect.

b) *Edge Detection and Highlighting:* Adaptive thresholding is used to detect and accentuate the edges within the image.

c) *Blending of Smoothed Image and Edges:* The smoothened image and the highlighted edges are then blended to create the final cartoon-like appearance.

By pressing the '0' key, users can activate the cartooning effect on the video stream, experiencing the transformation of real-time images into stylized, cartoon-like visuals.

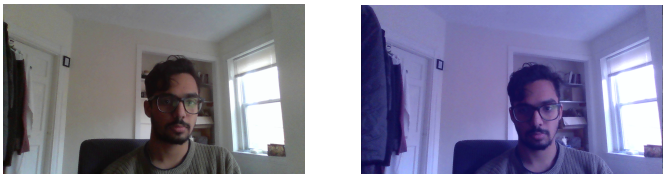

Fig. 14. Comparison of a Normal Video Frame (left) and a Brightened Video Frame (right)

Fig. 15. Cartoonize

V. Feature Detection and Matching

Feature Detection and Matching are critical components in Simultaneous Localization and Mapping (SLAM), a process used in robotics and augmented reality to build up a map of an unknown environment while simultaneously keeping track of an agent's location within it. Here's a brief overview of their roles in SLAM:

A. Feature Detection

Purpose: In SLAM, feature detection involves identifying distinctive points or areas in images that can be reliably tracked across multiple frames. These features should be invariant to changes in scale, orientation, and illumination to ensure robustness.

Common Methods: Popular feature detectors include Harris Corner Detector, FAST (Features from Accelerated Segment Test), and SIFT (Scale-Invariant Feature Transform). These algorithms identify points in the image that have a unique structure, such as corners or blobs.

In SLAM: Detected features are used to estimate the motion of the camera (or robot) and build a 3D map of the environment. In a dynamic setting, feature detection is continuously performed on new incoming frames.

B. Feature Matching

Purpose: Once features are detected, the next step is to match these features across different frames or different viewpoints. This process helps in determining how features (and thus the camera or robot) have moved relative to the environment.

Common Methods: Feature matching can be achieved through methods like Brute-Force matching and FLANN (Fast Library for Approximate Nearest Neighbors) which are often used to describe the features for matching.

In SLAM: Matching provides the correspondences needed to estimate the relative motion between frames (odometry) and to fuse data into a coherent map. Accurate matching is crucial for minimizing drift in the estimated trajectory and map.

C. Harris Corner Feature Detector with Brute-Force Feature Matching

The Harris Corner Feature Detector was implemented from scratch, showcasing the detailed process involved in edge detection within frames. The Harris method is known for its computational intensity, attributable to several intricate steps in the algorithm. The implementation encompassed the following key stages:

1) **Image Smoothing:** Initially, the image is smoothed to reduce noise and artifacts that could lead to false corner detections.
2) **Gradient Computation:** The gradients in the X and Y directions are calculated, which are essential in identifying changes in intensity that signify corners.
3) **Calculating Gradient Products:** The squared gradients (dx^2, dy^2) and the product of gradients ($dxdy$) are computed for each pixel, forming the basis for the structure tensor.
4) **Structure Tensor Formation:** A structure tensor of second moments is created for each pixel, encapsulating the local gradient information.
5) **Corner Response Calculation:** Finally, the corner response for each pixel is computed, which determines whether a pixel is part of a corner based on the eigenvalues of the structure tensor.

A novel approach for feature matching was employed in the project, focusing on the intensity of pixels within a 5x5 window in both the reference and target frames. The method involves the following steps:

1) **Intensity Measurement:** For each key point detected by the Harris Corner Feature Detector in the reference frame, a 5x5 window is defined around the point. In the target frame, a corresponding 5x5 window is defined and moved across the entire target image to find the best match.
2) **L1 Norm Calculation:** The L1 norm, or the Manhattan distance, is computed to quantify the difference between the intensities of corresponding pixels in these windows.
3) **Best Match Identification:** The point in the target frame that yields the lowest score (i.e., the smallest L1 norm) is identified as the best match for the key point in the reference frame.

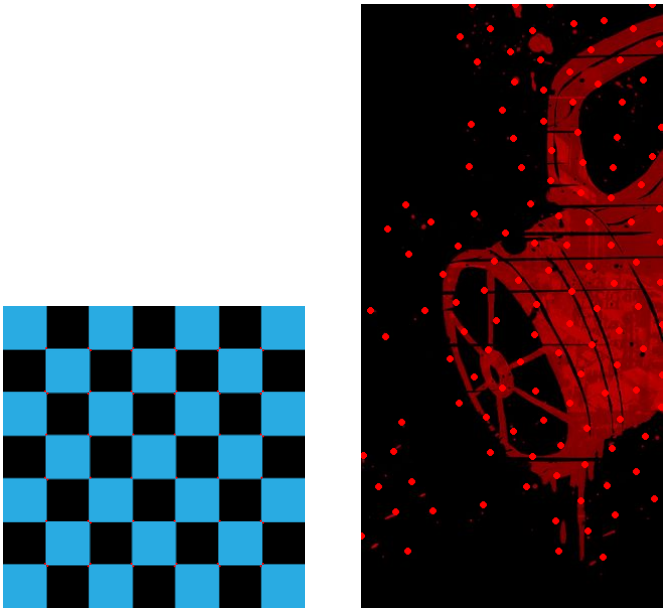

Fig. 16. Checker Board Corners (left) and Random image Corners (right) Corners marked in red

This method leverages the local pixel intensity patterns to establish an effective and straightforward mechanism for feature matching.

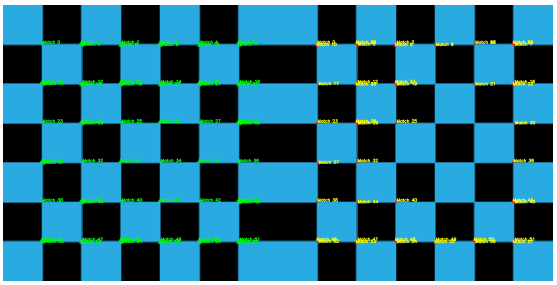

Fig. 17. Checker board feature matching

Adjustments were made to the minimum distance between features, threshold values and the number of required features to match in order to reproduce the results shown in the figures.

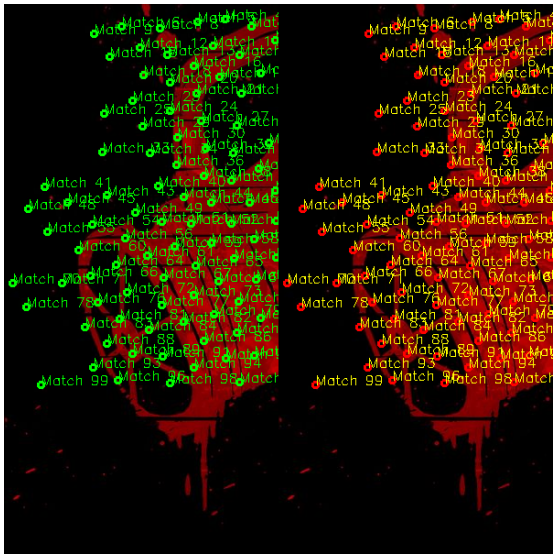

Fig. 18. Random Image feature matching

VI. References

[1] https://medium.com/@deepanshut041/introduction-to-harris-corner-detector-32a88850b3f6.

[2] https://docs.opencv.org/.

[3] https://ai.stanford.edu/ syyeung/cvweb/tutorial1.

[4] https://medium.com/@deepanshut041/introduction-to-feature-detection-and-matching-65e27179885d.

VII. Common Questions

1. What's the difference between OpenCV gray scale and custom gray scale?

OpenCV's standard grayscale conversion uses the formula $0.299R + 0.587G + 0.114B$. In contrast, my approach subtracts 255 from the blue channel, effectively inverting it. This inverted blue channel intensity is then applied to all BGR channels of the destination image, resulting in a unique grayscale effect.

2. In sepina tone filter how did I ensure to use original RGB values in computation?

A new image was created to ensure that the source image remains unaltered. For each pixel in the source image, RGB values were calculated and then updated in the new image. This approach ensures the originality of the source image while allowing modifications in the newly created image.

3. In Blur Filter what's the time taken by using the convolution approach and separate filter approach?

The normal convolution with a 5x5 Gaussian kernel took approximately 0.34 seconds for 10 iterations of blurring, whereas the separated filter approach, using a one-dimensional kernel and its transpose, significantly reduced the time to just 0.0034 seconds for the same task.

4. What operating system did I use and what's the IDE?

I used Linux operating system and Visual Studio code is the IDE that I used.