CS 5330 Pattern Recognition and Computer Vision
PROJECT-1

Ruchik Jani NUID - 002825482
Anuj Patel NUID - 002874710

1) A short description of the overall project in your own words.

The project appears to be a real-time video processing application using OpenCV in C++. The main functionality includes capturing live video from a camera, applying various image processing filters, and displaying the results in a window. We can interact with the application by pressing different keys to trigger specific filters or effects. The implemented filters include grayscale conversion, a custom grayscale filter, a sepia tone filter, 3x3 Sobel X and Sobel Y filters, face detection, an embossing effect, and adjustments for brightness and contrast.

The code is organised into different files, with `vidDisp.cpp` containing the main application logic, and `filters.cpp` containing the implementations of various image processing filters. The project uses classes and functions from the OpenCV library for image processing tasks.

The application provides a user-friendly interface for real-time experimentation with different image processing techniques on live video. Users can switch between filters and effects by pressing specific keys, allowing for an interactive and dynamic visual experience. Overall, the project showcases the capabilities of OpenCV for real-time computer vision applications.

2) Any required images or text along with a short description of the meaning of each image or diagram.

1. Display grayscale live video

(This is the image for comparison with the other outputs.)

For the 'g' key, the code converts the video frame to grayscale using the cvtcolor function.
The cvtcolor function in OpenCV is a versatile tool used for colour space conversions in image processing. It stands for "convert colour" and allows us to transform an image from one colour space to another.

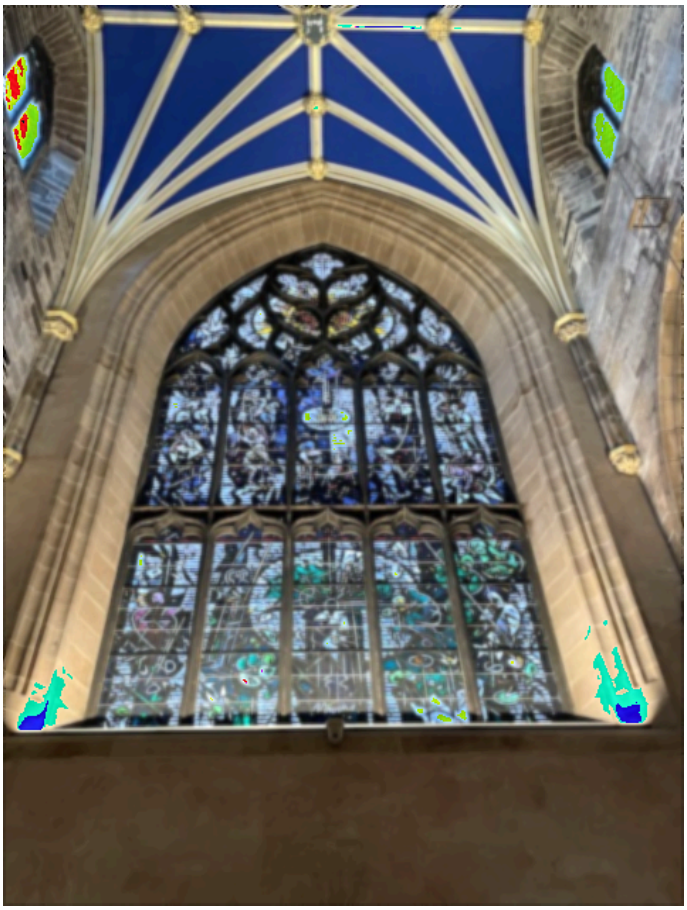
2. Display alternative greyscale live video.

When the user presses the 'h' key, the greyconvert class is utilised to apply a custom grayscale filter to the video frame, and the modified image is displayed in real-time. The getDestinationImage function ensures that the original image data is not modified during the filtering process.

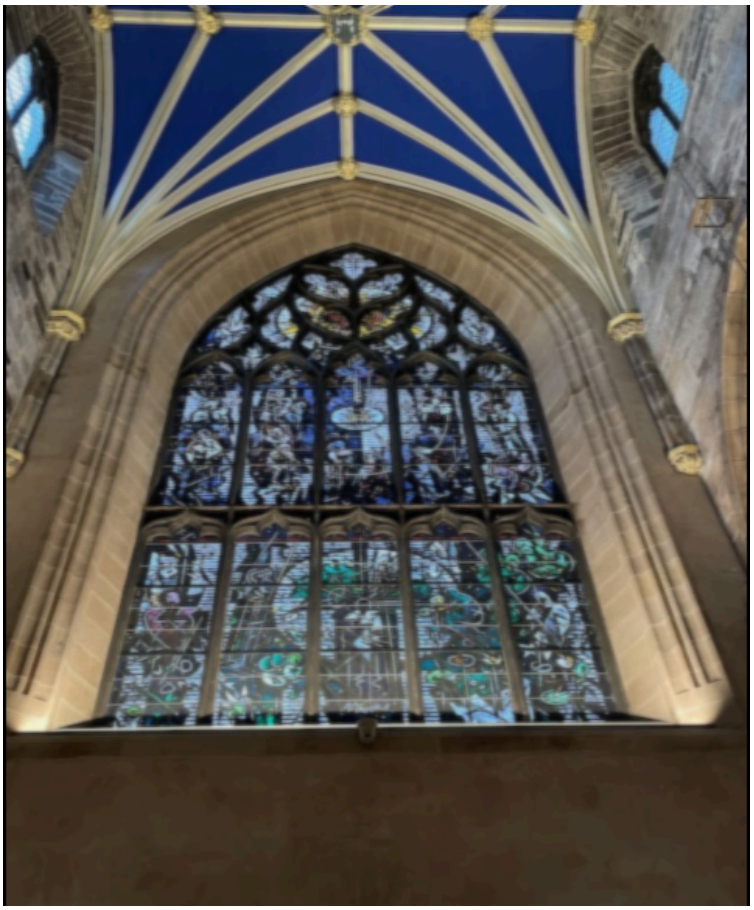
3. Implement a Sepia tone filter

The provided code applies a sepia tone filter to an image using the specified matrix coefficients: 0.272, 0.534, 0.131 for the blue channel, 0.349, 0.686, 0.168 for the green channel, and 0.393, 0.769, 0.189 for the red channel. These coefficients determine the transformation of each color channel to achieve the desired antique camera effect. The implementation ensures the integrity of the original RGB values in the computation, and the resulting sepia tone filter is visually presented in the report. Additionally, the code can be extended to incorporate vignetting, gradually darkening the image towards its edges, enhancing the overall vintage aesthetic.

4. Implement a 5x5 blur filter:



Blur5x5_1

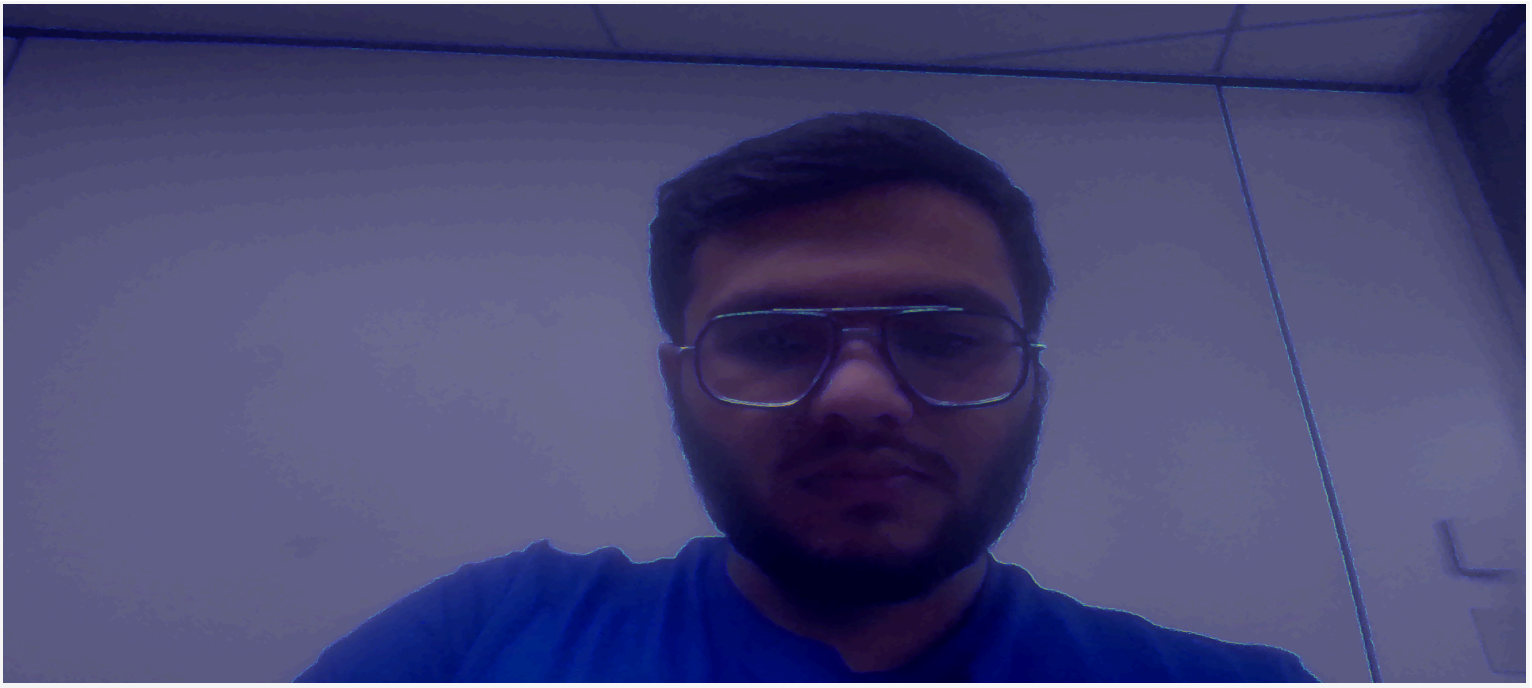


blur5x5_2

Time per image (blur5x5_1): 0.0956 seconds
Time per image (blur5x5_2): 0.0371 seconds

The output indicates that the blur5*5_2 function performs significantly faster than the blur5*5_1 function. The time per image for blur5*5_1 is measured at 0.0956 seconds, while the time per image for the optimised blur5*5_1 is notably quicker at 0.0371 seconds. This substantial time difference is attributed to the algorithmic differences and optimizations implemented in the two functions. The blur5*5_2 function employs a more efficient approach by using separable 1x5 filters for horizontal and vertical directions, reducing the computational complexity compared to the nested loop structure in blur5*5_1. These time measurements highlight the practical impact of algorithmic optimizations, underscoring the importance of such improvements for achieving better performance in image processing tasks.

5. Implement a function that generates a gradient magnitude image from the X and Y Sobel images



The `magnitude` function , computes the gradient magnitude image from X and Y Sobel images. It takes these images as input, calculates the Euclidean distance, and produces a normalised uchar colour image suitable for display. When triggered by the 'm' key in the `vidDisplay` program, it overlays the original video with the gradient magnitude visualisation, enhancing the understanding of image gradients.

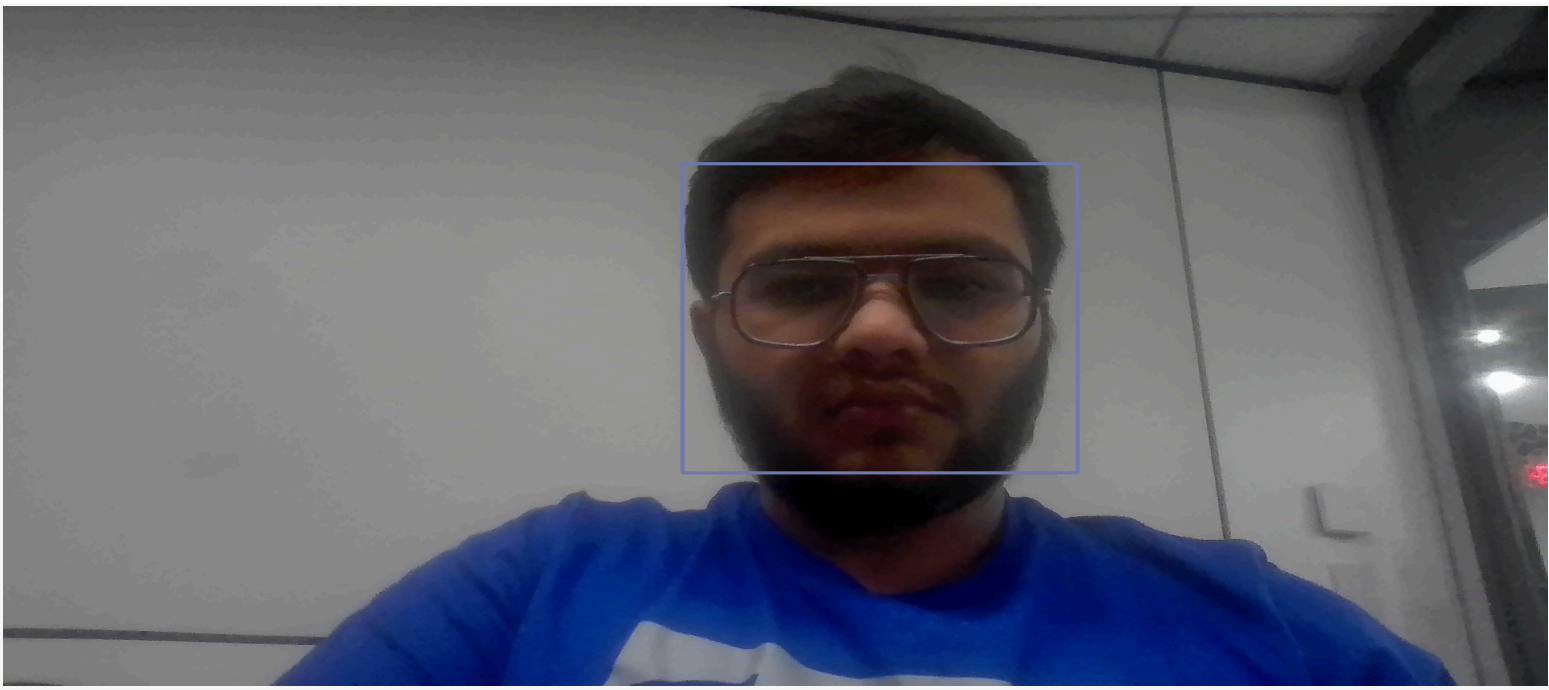
6. Implement a function that blurs and quantized a colour image

Original Image

Blurred & Quantized Image

This code block checks if the user pressed the 'l' key. If true, it applies a blur and quantization filter to the video frame with a quantization level of 10. The modified frame is then assigned back to the original video frame variable. Finally, the updated video frame is displayed. The exact workings of the blur and quantization processes are encapsulated in the blurQuantize function.

7. Detect faces in an image

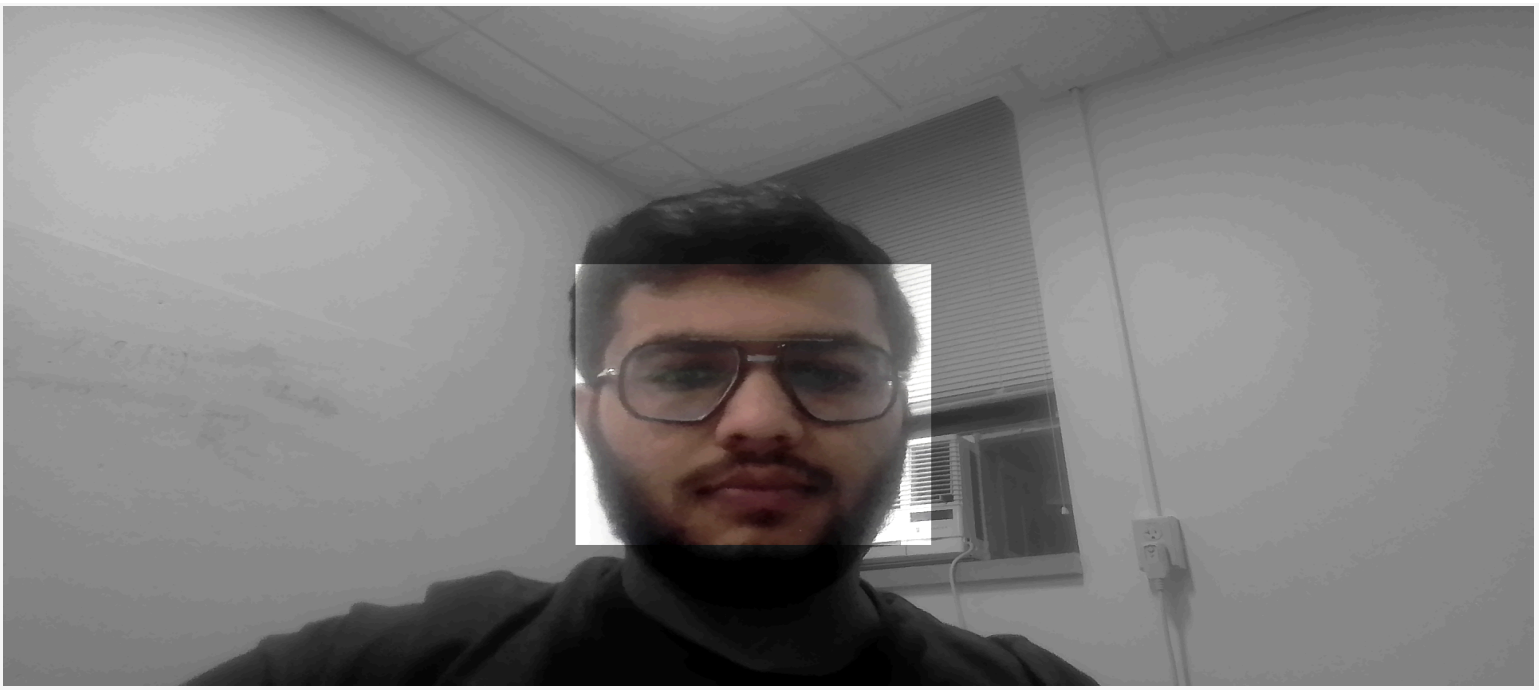


The face detection is based on the Haar cascade classifier, and the code involves integrating the face detection functions into the video stream program, responding to user input ('f') to enable face detection. Detected faces are then highlighted with rectangles in the displayed video frame.

8. Make an embossing effect (hint, use the Sobel X and Sobel Y signed outputs and take a dot product with a selected direction like (0.7071, 0.7071)

The embossing effect is achieved by applying Sobel X and Sobel Y filters to the input image. These filters detect the intensity gradients in the horizontal and vertical directions, respectively. The resulting gradients are then combined to calculate an embossing value for each pixel, considering a specified direction. This value is adjusted to be within a valid intensity range and is used to set the pixel values in the destination image. The key idea is that the embossing effect highlights the edges and enhances the three-dimensional appearance of the image by simulating the effect of directional lighting. The direction of the embossing effect can be adjusted by specifying the weights (`directionX` and `directionY`) for the Sobel X and Sobel Y gradients. The final output is a visually interesting image with enhanced texture and depth, resembling an embossed or engraved appearance.

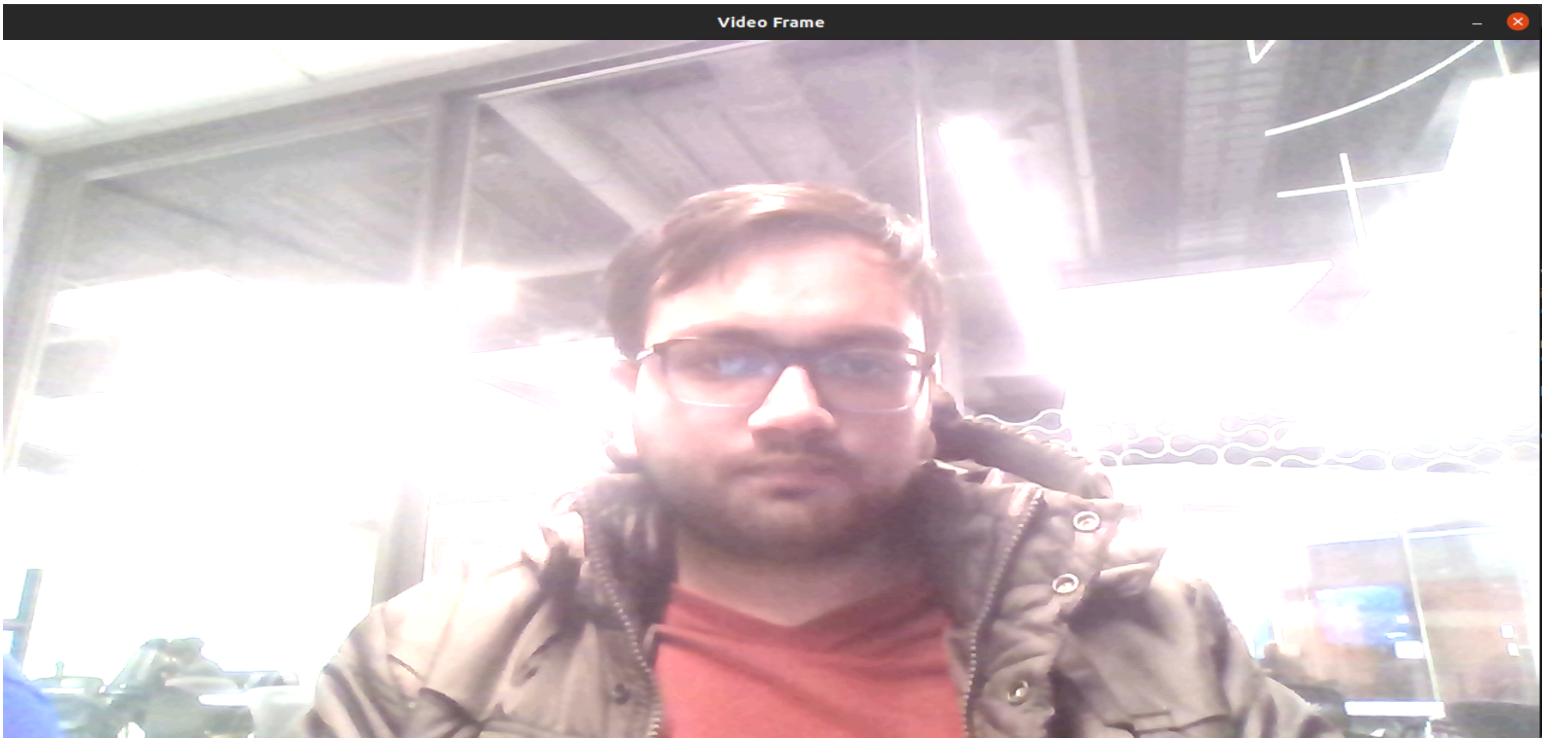
9. Make the face colourful, while the rest of the image is grayscale



The process of making the face colourful while rendering the rest of the image in grayscale involves several steps. Initially, the program captures video frames from the camera feed and processes each frame in real-time. Upon detecting faces using the implemented face detection algorithm, the code extracts the region of interest corresponding to each detected face. This region is then converted to greyscale, creating a desaturated representation of the face. The greyscale face is transformed back to the BGR colour format, preserving its original colour information. Finally, the modified grayscale face is seamlessly integrated back into the original frame, effectively rendering the detected faces in colour while the surrounding areas remain in greyscale. This approach provides a visually appealing effect, accentuating the facial features against a subdued background, and demonstrates the capability of the computer vision system to selectively manipulate colour information in real-time video processing.

10. Allow the user to adjust brightness or contrast.

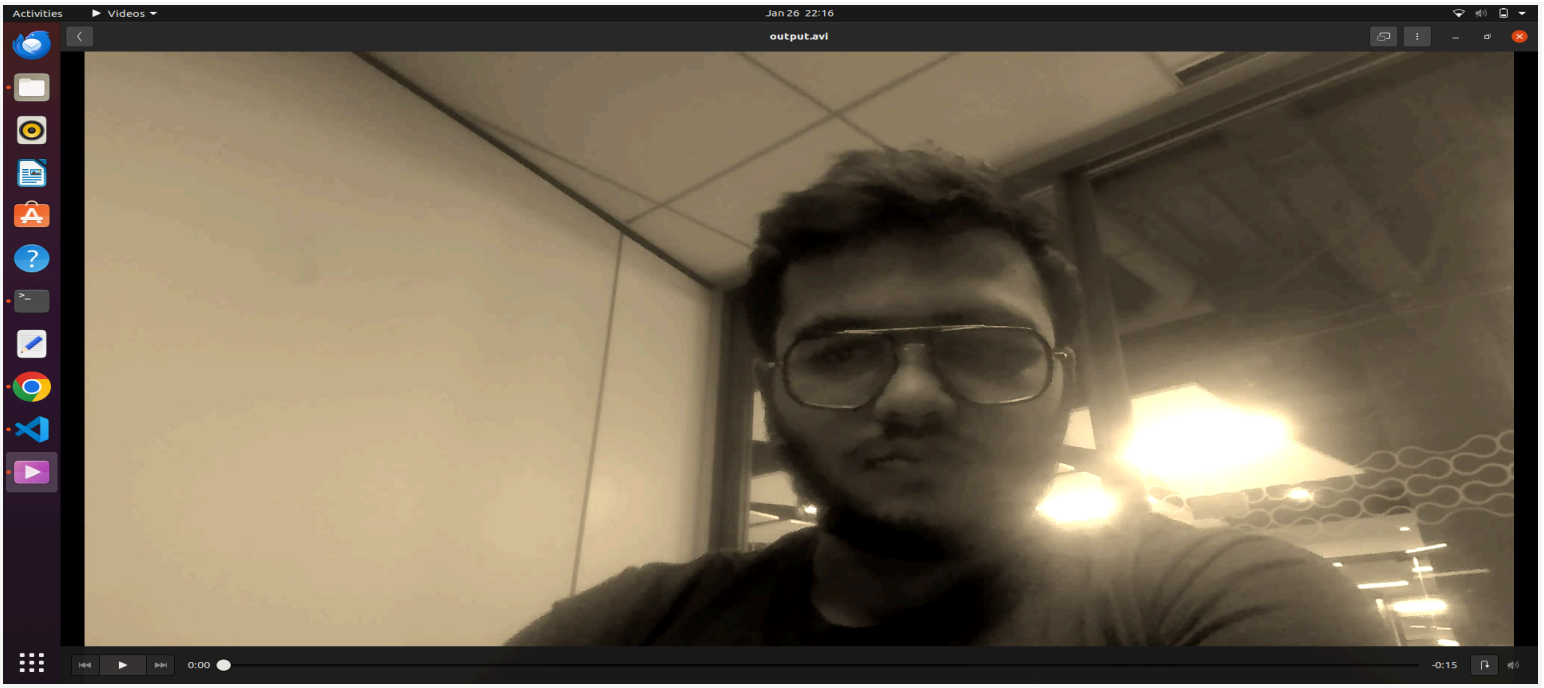
Original Image



Brightness (beta) - 50, Contrast (alpha) - 1.5

In the C++ code, brightness and contrast adjustment is handled by the `adjustBrightnessAndContrast` function. It takes a `Mat` image, along with contrast (`alpha`) and brightness (`beta`) parameters. The function performs accurate arithmetic by converting the image to `CV_32F`, applying alpha and beta, and then saturating values to a valid range. The image is finally converted back to `CV_8U`. To adjust brightness and contrast in the video feed, call this function with desired alpha and beta values.

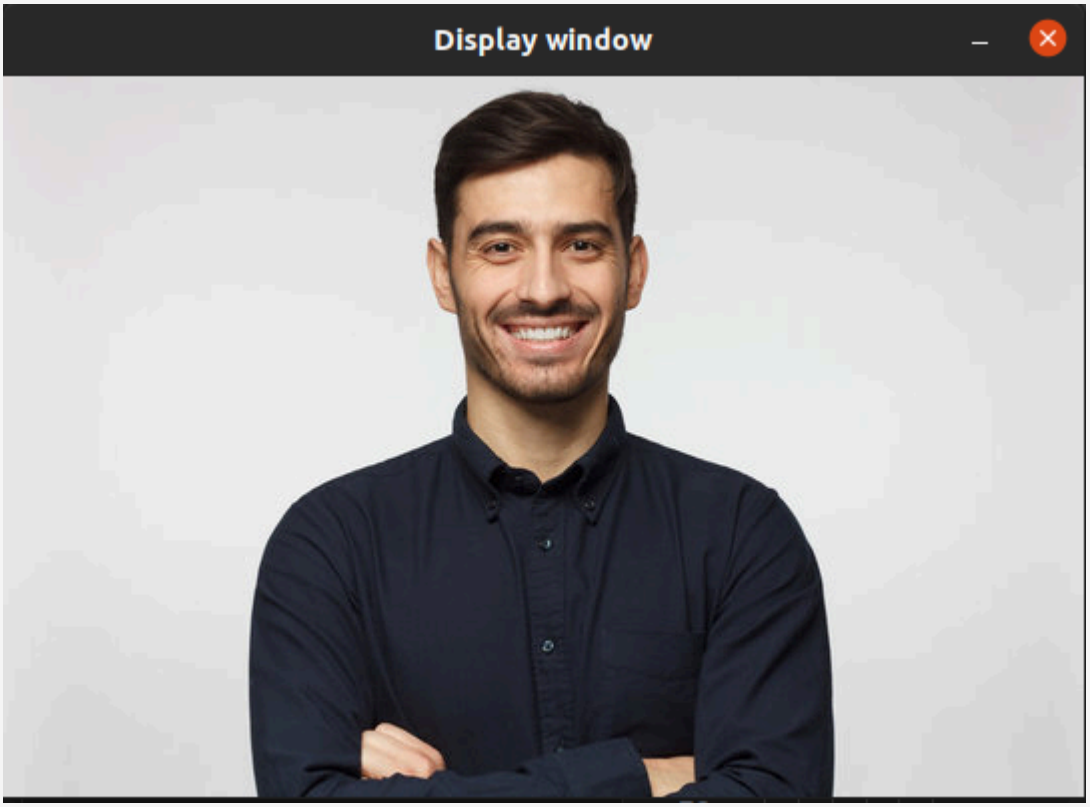
11. Let the user save short video sequences with the special effects.(extension)



The 'v' key in the program functions as a toggle switch for video recording. It flips the `saveFrame` flag, enabling or disabling video recording. When active, each camera frame is captured and saved to a video file. using the specified `VideoWriter`. This simple yet effective implementation allows manual control over video recording, making it easy to start and stop recording with a key press. The provided video is recorded while

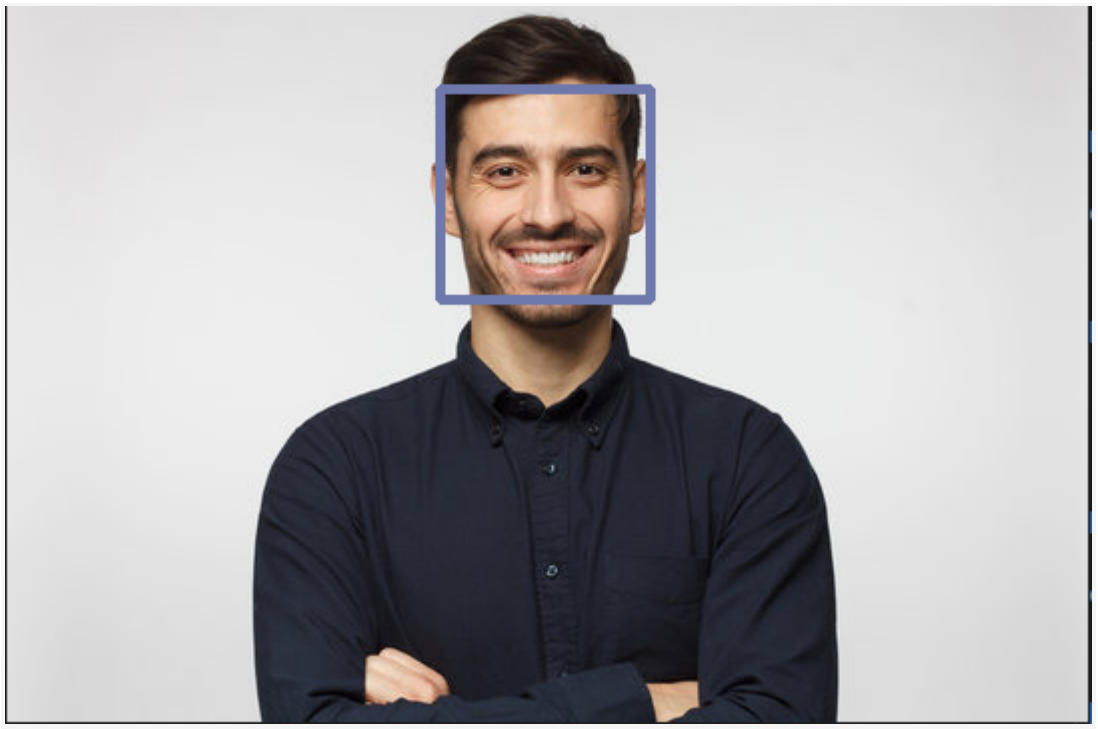
Pressing the key v.
output.avi

12. Implement your effects for still images and enable the user to save the modified images. (extension)



Original Image

Greyscale Image



Face Detection

Sepia FIlter

On the original image, we have applied the greyscale filter ('g' key) which converts the original image to greyscale using OpenCV's cvtColor function. Secondly, the sepia filter ('i' key) is applied by utilizing a custom sepiafilter function, transforming the original image into a sepia-toned version. Lastly, the face detection extension ('f' key) involves converting the image to greyscale, detecting faces using a dedicated function (detectFaces), drawing bounding boxes around the identified faces, and displaying the image with the added visual information. The implementation also includes a smoothing mechanism for the last detected face's position to enhance visual stability in consecutive frames.

13. Let the user add captions to images or video sequences when they are saved (i.e. create a meme generator). (extension)

In this extension, it is implemented to save modified images with user-added captions. Upon detecting the 's' keypress, the program prompts the user to input a caption. It then creates a copy of the current image, adds the user-provided caption using OpenCV's putText function, and saves the modified image as "Image_Saved.jpg". Finally, the program prints a confirmation message, indicating that the image has been successfully saved with the specified caption.

3) A short reflection of what you learned.

In this project, we enhanced our skills in C/C++ and OpenCV, focusing on image and video processing. We learned to implement various image transformations and filters, such as greyscale, sepia, and Sobel filtering, gaining a deep understanding of pixel-level image manipulation. Handling real-time video streams and developing interactive features that respond to user inputs were key aspects of our learning. This experience also improved our proficiency with software development tools and honed our problem-solving skills, particularly in optimising for performance. Overall, the project provided a comprehensive foundation in computer vision, blending theoretical knowledge with practical application.

4) Acknowledgement of any materials or people you consulted for the assignment.

1) https://opencv.org

2) https://makefiletutorial.com/

3) https://www.imageprovision.com/articles/understanding-image-filtering-techniques-in-image-processing#:~:text=Image%20filtering%20is%20a%20technique%20that%20is%20utilized,based%20on%20the%20values%20of%20the%20existing%20pixels.

4) https://www.google.com/

5) https://youtu.be/m9HBM1m_EMU?si=PJ2MVrtGrqTbhhNX