Video Special Effects

In this Project, I have created multiple video/image where the image frames are taken from a live video stream and also are read from a specific path mentioned from the command line. Below are the different special effects implemented.

1. **Read an image from the file and display it**: In this implementation, the image file is directly read from a specific path and is displayed.

2. **Display Live video**: In this implementation, the image frames are directly read from a live video stream and then processed to display each frame which gives a video effect. Hen pressed ‘q’, the display will terminate and the program ends.
3. **Display Grayscale live video:** In this implementation, I have used the cvtColor method of the cv library to get the grayscale image, when pressed ‘h’.

the 1st image is the normal image and the second is the greyscale version of the normal image which is obtained using cvtColor method.

4. **Display Alternate Grayscale live video:** In this implementation, I have created the grayscale image from scratch. I have summed all channels value and divided it by 6 to get the half of average value across all channels as grayscale value.

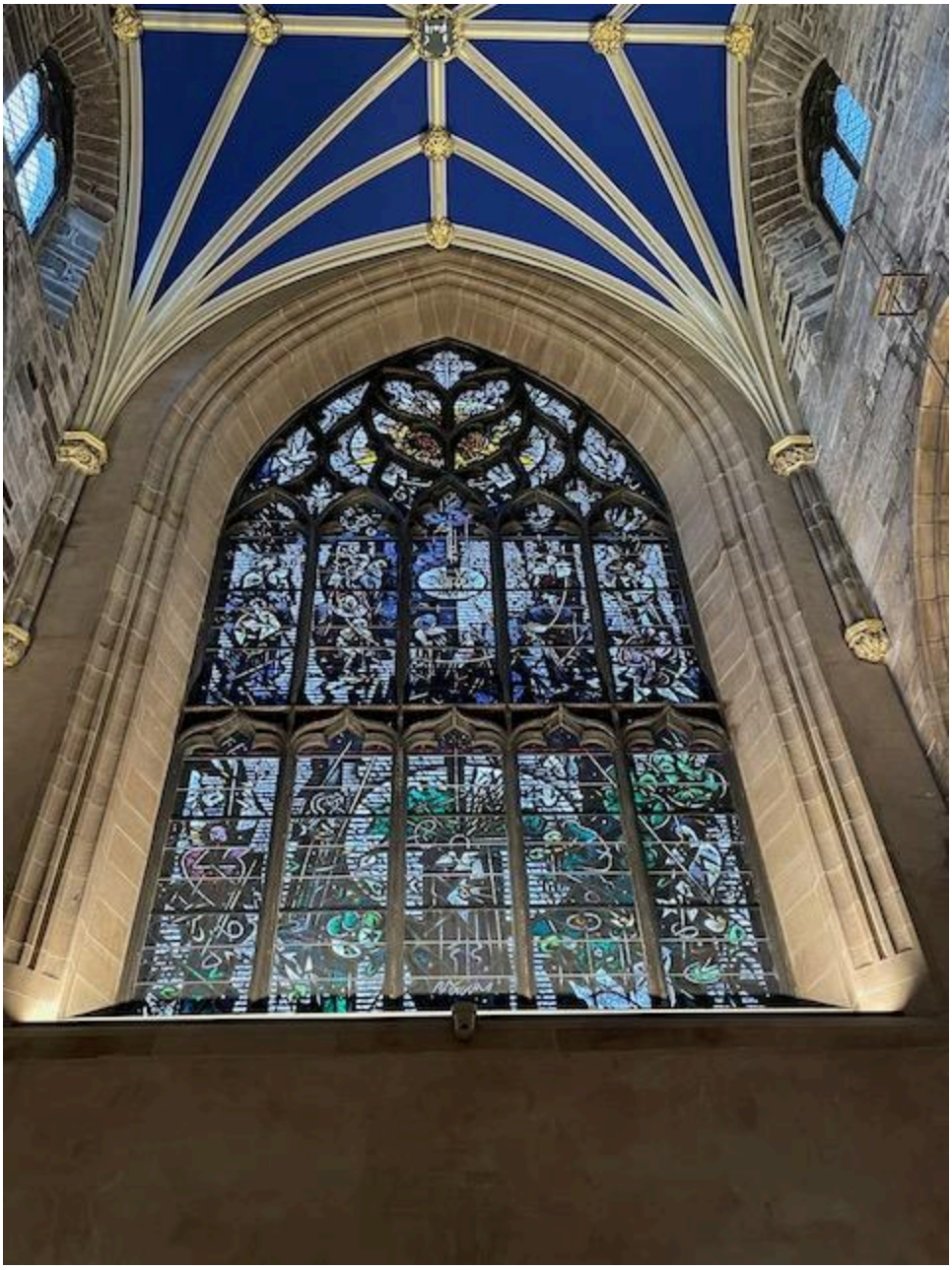
The 1st image is the normal image and the second is the alternate grayscale version of the normal image. We can see that the greyscale generated using cvtColor method and the one here is different. It's because the cvtColor method just chooses one channel and that's the greyscale value. However, in this implementation, I have taken average across all channel values and have reduced it by 2 and hence I get a darker image.

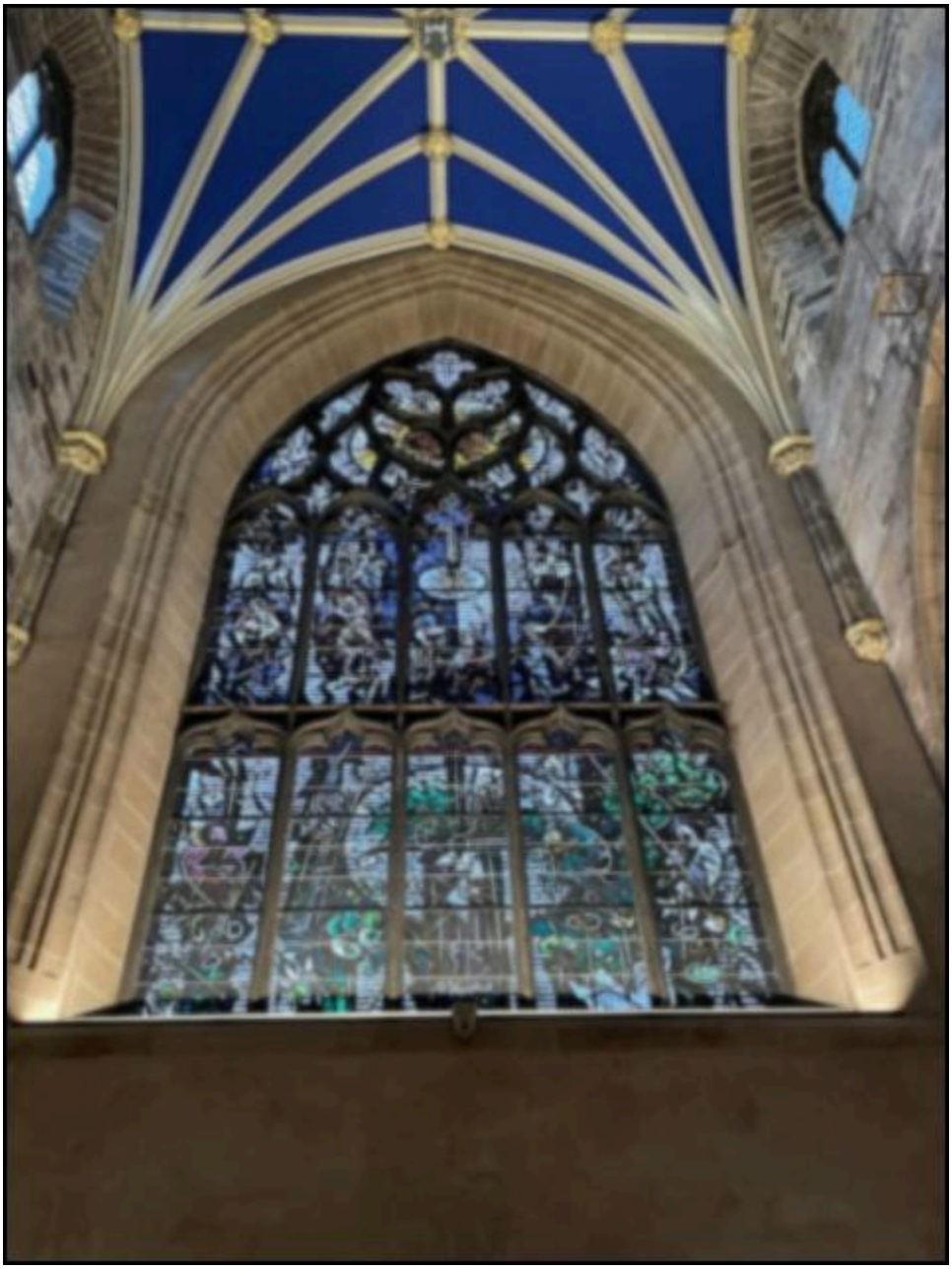
5. **Sepia Tone Filter:** In this implementation, we need to update each color channel value. Since each color channel value is needed in calculating the new values of color

channels, we use 3 temp variables to store the new values from the existing color channel values. Once we calculate all the 3 color channel values, then I update the channels with the new values.

The original image is present in section 1 and this is the result when we apply a sepia filter.

6. 5*5 Filter:
    a. In the first implementation, I have used a 5*5 Gaussian filter and have multiplied this kernel across the target pixel to get the updated value. For running this method to blur the given below image 10 times, it takes around 0.1716 seconds.
    b. In the second implementation I have used a 1-D array of size 5 to get the same results as above and is much faster. It uses a kernel multiplier to get the kernel values and computes the blur value for each pixel. I have also not used the ‘at’ method to access each pixel value. Instead, I have used a pointer to access pixel values and it is much faster. For running this method to blur the given below image 10 times, it takes around 0.1060 seconds. We can clearly see that this is much faster than the previous implementation.





The 1st photo is the original image and the second is the blur image obtained after applying a Gaussian filter.

```
(base) sachinpc@Sachins-MacBook-Pro build % ./MyProject /Users/sachinpc/Documents/cvphotos/14.jpeg
Time per image (1): 0.1716 seconds
Time per image (2): 0.1060 seconds
```

7. **3*3 SobelX and 3*3 SobelY Filters as separable 1*3 filters**: In this implementation, I have used 1*3 separate kernel vectors for SobelX and SobelY filters. It uses this1*3 kernel along with a kernel multiplier to calculate the new pixel values at each point.

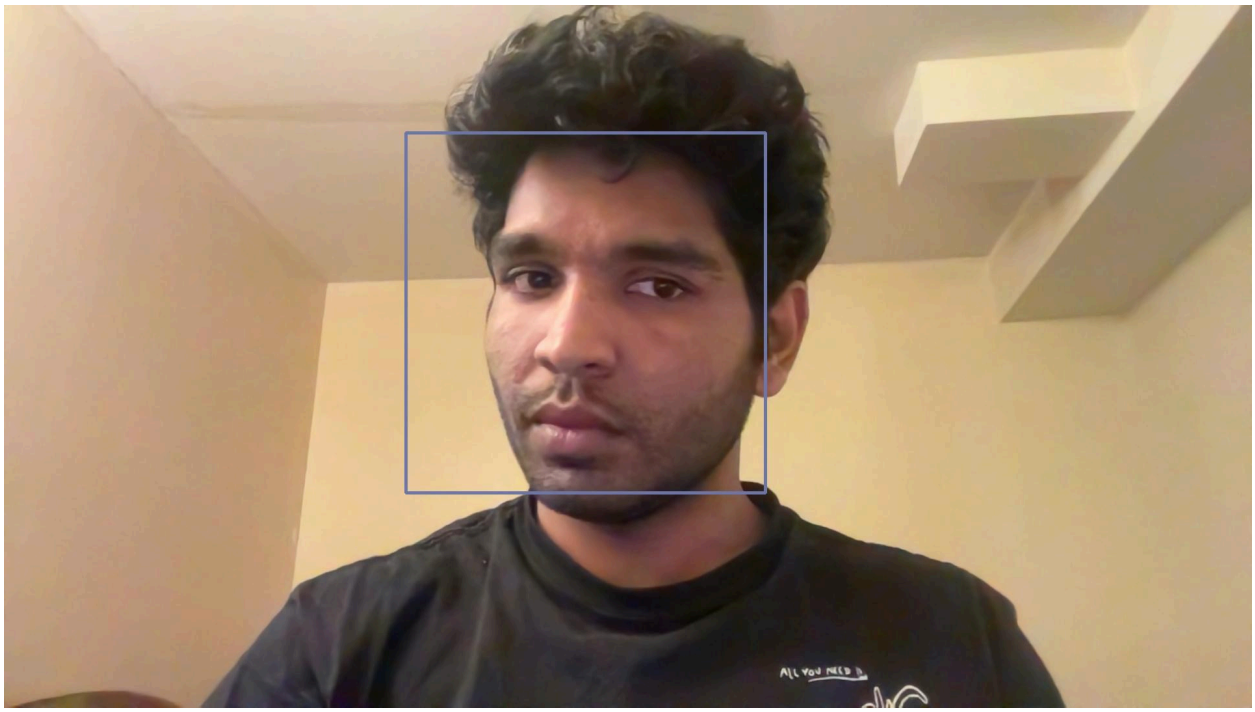
The 1st Image is SobelX image and the second image is SobelY image. We can clearly see that SobelX is able to determine vertical edges and SobelY filter is able to determine horizontal edges.

8. Gradient magnitude image from the X Sobel and Y Sobel images: I have taken the outputs of SobelX filter and SobelY filter and then computed euclidean distance between SobelX and SobelY filter at each point and that gives us the magnitude image combining

horizontal and vertical edge.

9. **Blurs and Quantizes a color image:** In this , we select a level and then divide the max intensity with each channel by level.Divide the value obtained by the pixel value and later multiply by the bucket value. This will give the quantized value for that image.

10. **Detecting faces in an image:** We use faceCascade to detect the faces in the given image and then use a rectangle to draw a bounding box for the faces.



Additional Effects Implemented

1. **Adjust Brightness and Contrast of an Image:** The bright command is taken as an input from the command line and then, the user is asked the brightness intensity level. The user enters the value(positive implies the image is brightened and negative pimples the image is contrasted) and that value is being used to brighten or contrast the image. Basically, we increase or decrease the current value with the given value to get the new image. The user can enter any value to brighten or contrast the image. If the value exceeds the range, it's capped to the given range of values.

The 1st image is brightened by 100 and the second image is brightened by -100, i.e contrasted by 100.

2. **Stretch or warp the image:** For the given image, I'm stretching the image horizontally. I'm taking the columns and starting from the center column, for each side of the center column, I create 3 buckets. For all the columns which lie in the first bucket, the resultant image will contain those columns only once. The columns in 2nd bucket will be repeated twice and similarly, the columns in the 3rd bucket will be repeated thrice. Similarly done on the other side of the center column and the end result will give us a stretched image.

The 1st image is the original image and the 2nd is the stretched image. The stretched image is double the size of the input image.

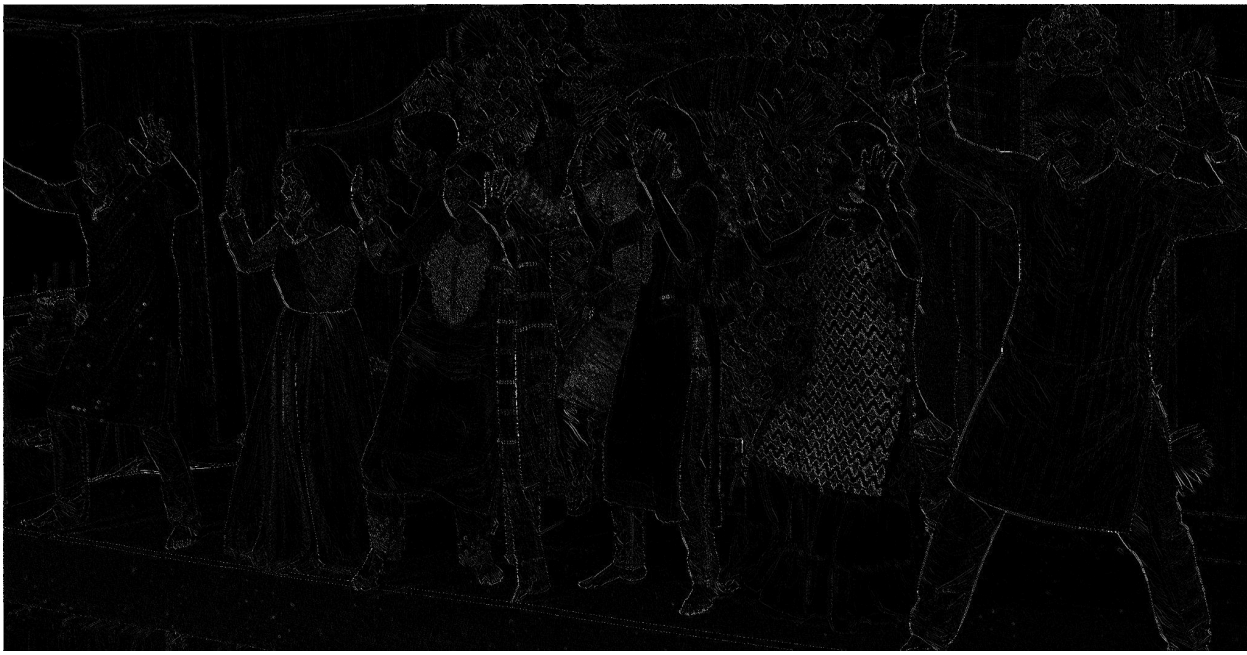
3. **Negative of the Image**: the negative of a given image is obtained by subtracting the each channel value by its maximum intensity and updating it as the new

value of that color channel.

4. **Color the Face and make the remaining part of the image Grayscale:** In this, I first detect the faces using faceCascade and then based on the bounding boxes of the faces, all the remaining regoing is made greyscale.

Extensions:

1. **Canny Edge Detection from Scratch:** Canny edge detection is used to detect the edges in a given image. This uses SobelX and Sobel Y filters in calculating the edges.
    a. The first step is to apply Guassian blur to smoothen the given image. Then Sobel X and Sobel Y are calculated.
    b. Sobel magnitude is calculated from SobelX and SobelY along with the gradient angle for each pixel value. The gradient angle value helps in detecting the direction of the edge.
    The gradient angle is obtained by taking tan inverse of SobelX over Sobel Y. Angle in radiance = tan-1(Sx/Sy), where Sx is Sobel X value and Sy is Sobel Y value.
    c. Now, we apply non Maxima Suppression for each pixel value. We obtain the pixel's gradient angle and then check if the current pixel value is smaller than any of its neighbors in the angle direction. If it is, we assign value 0 to it. Else, if it's greater than its neighbor pixel values, we keep the value as it is.
    d. After non Maxima suppression, we apply hysteresis thresholding. In this step, we decided on lower threshold and upper threshold values of pixels. All the values which are lesser than the lower threshold are assigned value 0. All the values greater than threshold are assigned value 255(white). Now, for all the values lying between lower threshold and upper threshold, we assign 255(white) to it, only if it is connected to a pixel which is greater than upper threshold, else its assigned 0(black). At the end of this step, we will have the image with significant edges only.



The first image is the soble magnitude image. The second image is the image obtained after doing non maxima suppression. The third image is the final end result after applying hysteresis thresholding. We can see that only the major edges are visible in the end result.

2. **Zooming only the face and leaving the background the same**: In this filter, we only apply zooming to the face detected and the remaining area is as it is.

3.