CS 5330 Project-1

Saideep Arikontham

Title: Exploring Video Filters and Real-Time Effects Using OpenCV

Project Overview

This project involved understanding and implementing various video filters and effects in real-time using OpenCV. It aimed to enhance video processing skills by creating filters such as grayscale transformations, sepia tones, Gaussian blurs, and Sobel edge detection. Additionally, the project explored tasks like detecting faces in a frame, playing around with depth information to modify video/frames. I divided the filters into two categories, one which applies its effect and alters the video stream directly and the other is where it only applies filters to one selected frame. Key press is most vital for this project to change between filter effect, apply certain filters to the selected frame, terminate the operation, save frames with video filters applied and more. The following are the list of effects implemented.

Filters altering video stream:

- Press 'c' to get the original video
- Press 'g' to change video to gray scale
- Press 'h' to change video to another gray scale
- Press 'a' to change video to sepia tone
- Press 'b' to blur the video (separable kernel)
- Press 'f' to get the face box for the video
- Press 'u' to get the mirrored video
- Press 'd' to get the portrait image (background blurred)
- Press 'o' to get the frame with fog effect
- Press 'k' to get the video with passing circle (***extention***)

Filter altering only the selected frame:

- Press 'x' to get the sobel filter applied frame
- Press 'y' to get the sobel filter applied frame
- Press 'm' to get the magnitude of sobel filter applied frame
- Press 'l' to get the quantized image
- Press 'p' to get the frame with sketch effect (***extention***)
- Press 'i' to get the frame with median filter

These implementations showcased the versatility of OpenCV in creating visually impactful and computationally efficient filters.

Directed Tasks and required images

1. **Greyscale filter (Req. image 1):** The image showcasing *greyscale filter (press ‘g’)* implemented using cvtColor directly.


2. **Different gray filter (Req. image 2):** The image shows my own version of implementing a *different gray filter (press ‘h’)*. For each pixel, I have taken the difference between maximum of the 3 channels (BGR) and minimum of the 3 channels divided by 2 to get one single value. I assign that resultant value to each of the 3 channels giving a different kind of gray filter. This filter is on the darker side compared to opencv’s grayscale filter and also retains less details like shadows compared to opencv’s grayscale filter.

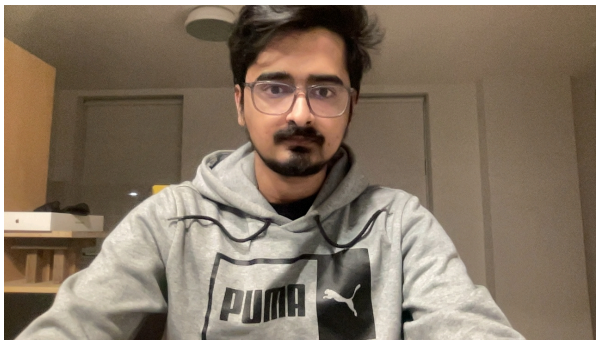

3. **Sepia filter (Req. image 3):** The image shows the implementation of *sepia filter (press ‘a’)* computed using the coefficients given. I retained the original values by assigning the computation to a temporary variable for each of the color channels. To ensure that the computed value lies in between 0 and 255, I used the minimum of the channel value and 255.

4. **Gaussian Blur (Req. Image 4):** The following is the result of using *Gaussian blur filter (press ‘b’)* with both 5x5 filter and, 1x5 and 5x1 separable filter. Out of the two, using a separable filter was faster as it reduces the total number of computations required. Below is the output of the separable blur filter. The intensity is not huge but the difference can be seen clearly near the hair (1st image is the original and 2nd image is the blurred version). The timing information is also included:

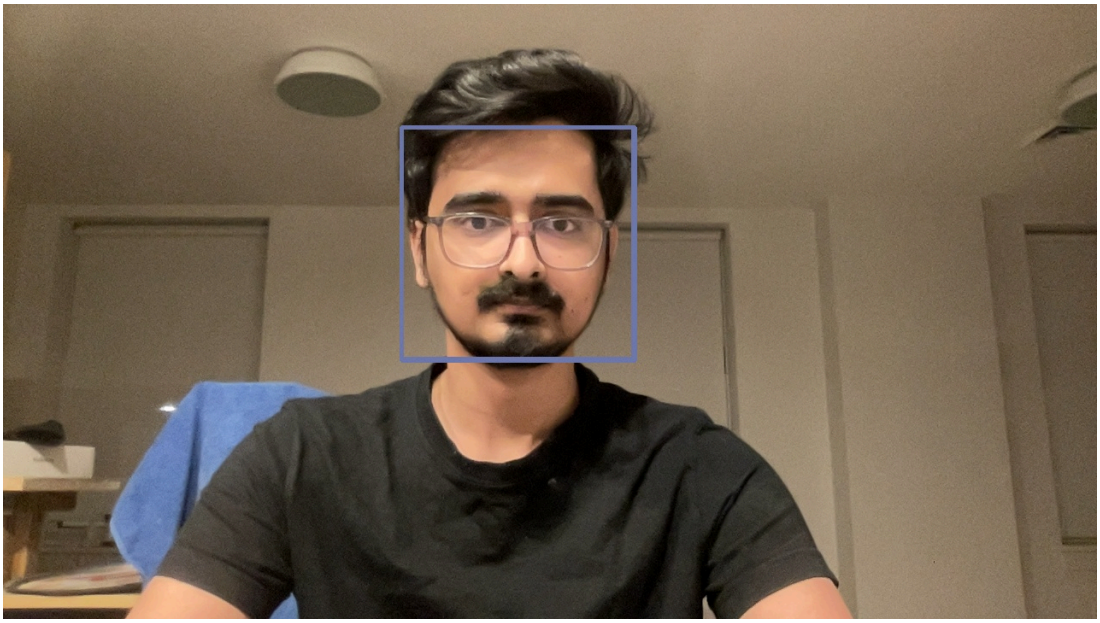
```
(base) saideepbunny@MacBookAir Exploring_OpenCV % ./bin/timeBlur ./images/cathedral.jpeg
Blur 1 results:
- Time per image (1): 0.0343 seconds
Blur 2 results:
- Time per image (2): 0.0199 seconds
```

5. **Magnitude (Req. image 5):** To generate the *magnitude (press 'm')* of an image, first Sobel X (3x3 kernel) and Sobel Y (3x3 kernel) are used. Sobel X (image 2 below) outputs vertical edges predominantly and Sobel Y (image 3 below) outputs horizontal edges predominantly. The magnitude (image 4 below) identifies edges of objects in the image which can be seen the output images (Specifically PUMA text of the hoodie) below:

6. **Blur quantization (Req. image 6):** This is the output of *blur quantization (press 'l')* with 10 levels. I used the blur5x5_2, the blur implementation using separable filters. Below is the output of using the filter.




7. **Face detection (Req. image 7):** This is the output of *face detection (press 'f')* in the video stream using the provided faceDetect.cpp. The given code resource puts a rectangular block around the detected face. This code has be integrated successfully into vidDisplay.cpp and the following output is obtained.



8. **Depth information (Req. image 8):** Using the given resources, I have successfully integrated the code to obtain depth information successfully. I have not included any functionality to display video depth alone. However, I have implemented to save the depth information image while saving any filtered frame that uses depth information. The following is the depth image of a frame followed by the original image and the background blur image (Required image 9). I have implemented the filter such that it is directly applied to the video stream. Since the processing time is more, the video can be

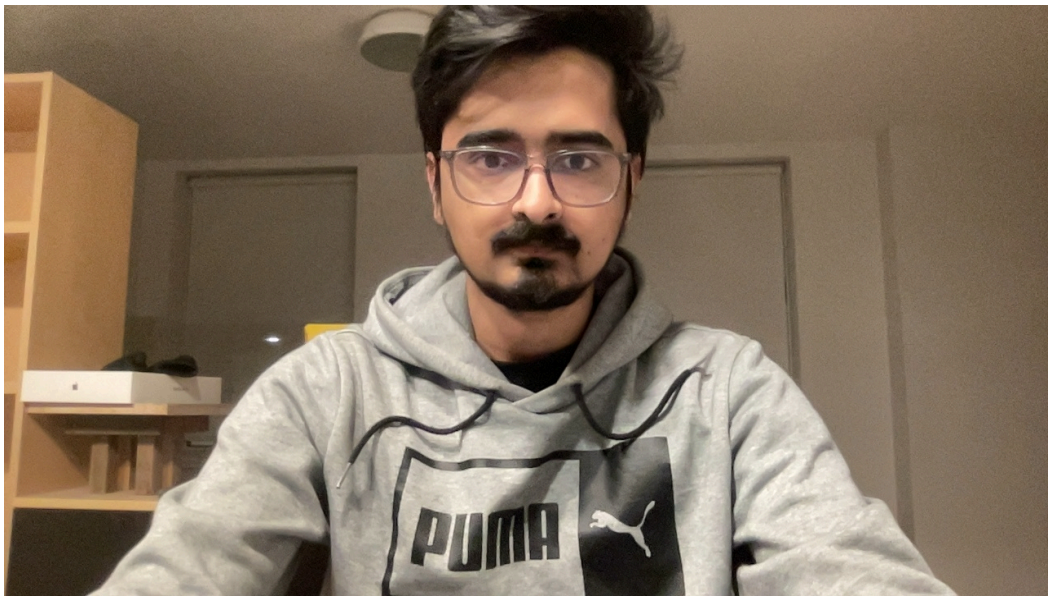
clearly seen to be outputting less fps. So, I have tried reducing the window size (still less fps.)

9. **Background blur (Req. image 9):** This filter implements a background blur (press ‘d’). The main idea is to use the depth information obtained from above to define what is background and what is foreground. This is done using thresholding i.e., depth values (ranging between 0 and 255) above 70 are set to white and depth values less than or equal to 70 are set to black. Then a duplicated frame is taken and high intensive blur is applied. Then pixels will be iterated to check if that pixel is part of the background or foreground. If its background, then the blurred pixel is written to output. If its foreground, then the original pixel is written to output, therefore giving a background blur effect.

Three more filters and required images.

1. **Mirror filter (Req. image 10):** This effect is obtained by modifying individual pixels (press ‘u’). This filter mirrors the video stream by taking each frame, interchanging the left side pixels with right side pixels. Below is the obtained output.

2. **Median filter (Req. image 11):** The median effect (press ‘i’) is obtained by modifying individual pixels using the area of filters around them (5x5). I used padding for this filter to account for the top, left, right and bottom frame. Then, for each of the 5x5 blocks, the median of each color channel’s value is assigned to the respective single pixel to obtain the effect of a median filter. This effect only modifies a single selected frame (when ‘i’ is pressed) and not the complete video stream. This filter seems to applying a different kind of blur which smooths the image.

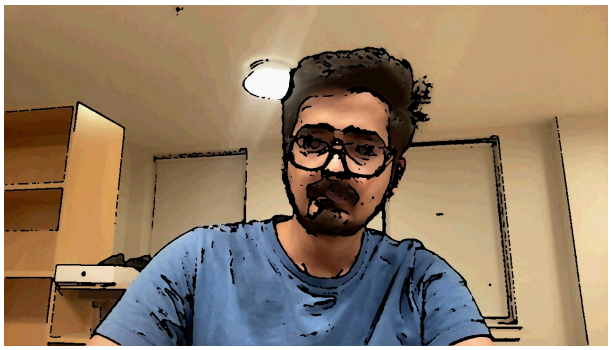


3. **Fog filter (Req image 12):** The fog filter (press ‘o’ key) applies a fog effect to the video stream. The fog effect will affect pixels that are far away with a higher intensity compared to the pixels that are closer to the camera. This is implemented with the help of depth values and using an exponential decay of depth values. I used a red tinted fog as most of the background was already white.


- The depth information has picked up additional pixels to be a part of me even though they are part of the background, this might be due to the reduced window size. However, we can see that the fog is being appropriately applied from the depth values given.

Extensions

1. **Sketch effect (press ‘p’):** The goal was to get an image which would look like a sketch rather than a normal image. Borders would be an important part of a sketch and one way to get them was using magnitude. Once the magnitude is obtained, threshold was used to convert black parts to white and white parts to black. Then, a medianBlur from opencv is used to smooth the output from previous operation (my median filter implementation did not work as intended) therefore it would look like a sketch drawn with only a marker. Now, to fill the color, I have taken the frame and applied blur quantization first. Then I have overlaid the sketch output over it so that the result would look like a colored sketch.

- The eyes were not as clear, which is because of the usage of magnitude but, the rest of the image definitely gave the effect I was looking for.
- This filter only picks one frame and applies this effect and does not alter the entire video stream.

2. **Passing circle effect:** The goal was to add a circle that would move from left to right on the screen. A simple circle object was added using opencv's function and its x coordinate is increased constantly. Once the width is reached, it will be reset to 0 using the modulus operator. This effect modifies the entire video stream.


Reflection

Through this project, I gained a deeper understanding of OpenCV's capabilities. It was really exciting and extremely satisfying to watch the effects work as intended. My understanding of color channels, pixel modifications, depth values and integrating different filters has definitely improved. The hands-on experience emphasized the importance of optimization techniques for real-time video processing, such as using separable filters for Gaussian blurs. I would still want to understand how it would be possible to integrate a seamless depth integration to video streams. There was a lot of experimentation to see what happens when we tweak certain parameters which is also a great learning. This is really a great way to be introduced to working on OpenCV and computer vision capabilities and I am looking forward to a lot of learning and exciting project work.

Acknowledgements

The best resources I found for this assignment were stackoverflow and ChatGPT. I used stackoverflow to understand certain errors, the requirement of temporary buffers and more. ChatGPT also helped me understand how certain OpenCV functions work internally, to explain stuff that I didn't understand immediately. One such example is the separable filters. I used ChatGPT to understand how the filter is separated which I was unable to understand prior to this. I wish I referred more to the official documentation but I found these two resources answering all my questions. Also, the clear direction given in the project requirements made it easier to understand and implement all the filters.