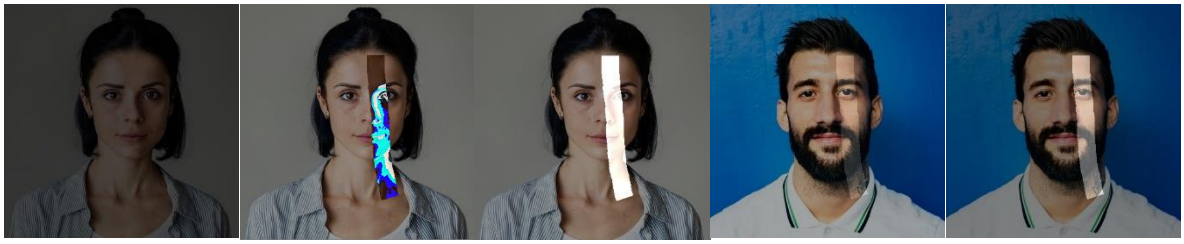
Student: rvkb79

Image Processing Report

Problem 1 – Light Leak and Rainbow Light Leak



Figures 1 - 5

Development

I started by implementing image darkening. See figure 1 with the darkening coefficient set to 0.7. I then drew a placeholder light leak mask in MS Paint, and wrote code to overlay it onto the input image.

This produced odd results, as seen in figure 2. I realised that some of the RGB values were overflowing when the intensity of the mask pixel plus the that of the corresponding image pixel totalled over 255. The uint8 data type that NumPy was using here can only store 256 values per pixel per channel, so it would loop back around to 0 when it exceeded the maximum. I fixed this issue in my code by setting any sum over 255 to equal 255. The result of this fix can be seen in figure 3.

I then added another parameter to control the blend between the two images. I interpreted the problem description as meaning this coefficient should control the brightness of both images at once (inversely). Since there is a separate darkness coefficient for the input image, I think the user would have better control of the blend between images if they had a darkness coefficient for the mask as well and no blend parameter. That would mean the input image would not be darkened twice, however I wanted to follow the task specification as closely as possible, so I did not implement it this way. face2.jpg is shown with darkness and blend coefficients 0.2 & 0.2 in figure 4 and 0 and 0.5 in figure 5, which has a stronger light leak, but the face is too dark.

Figure 6

Next, I tested my code with a placeholder rainbow mask, as seen in figure 6. This worked immediately, as my code already handled the colour channels separately.

My final step was to make some more realistic masks. I do not have any experience with image editing software, so I used shapes and gradient fill in PowerPoint before saving them as PNGs and loading them into MS Paint to resize them to 400x400 pixels. It is worth noting that my code does handle different dimensions of input image. My solution is to resize the mask with cv2.resize() so it covers the input image.

Figures 7 to 10 show the best parameters for each image and mode. In all of them, the darkness coefficient is 0.

Figure 7 – Blend = 0.4 *Figure 8 – Blend = 0.3* *Figure 9 – Blend = 0.25* *Figure 10 – Blend = 0.3*

Computational Complexity

My code features a triple-nested loop: for every column, for every pixel in that column, for every colour channel of that pixel, it performs branching and addition. However, the number of channels of an input image will never be above 4, so the innermost loop runs in constant time. Therefore, only two loops grow with input size, so it has time complexity $O(CR)$, where C and R are the dimensions of the image.

Problem 2 – Pencil / Charcoal Effect

Figure 11

Development

I generate the noise texture by using NumPy's Gaussian random number generator, which returns a grid with numbers varying above and below 0. The standard deviation of the numbers can be controlled by the noise parameter I have provided in my function; this ultimately controls how messy and strong the pencil strokes look. A visualisation of the noise overlayed onto a greyscale image1.jpg is shown in figure 11 (zoomed in).

Figure 12

Next, I added motion blur to the noise. In my first attempt, I convolved a mask around the image where the middle row of the mask contained all 1s, and the rest were 0s. This achieved a pencil stroke effect, albeit with very fine strokes (see figure 12, zoomed in for better clarity). Another issue was that the horizontal direction made it look too precise and uniform. I decided to make the strokes come in at a slight angle by distributing 1s in a diagonal across the mask. Figure 13 shows the result of this, which looks more natural, but the strokes are still too thin. My final improvement to the mask was to duplicate the diagonal line of 1s above and below the original, to create a thick diagonal line with a width of about 4 pixels. Now the strokes are broader and more obvious, as illustrated in figures 17 to 20.

Figure 13

The blend coefficient is applied only to the pixels of the noise texture before they are added to the image. Unlike in Problem 1, it doesn't make sense to darken the input image at the same time, because the overall intensity of the image should stay the same: pixels of the noise texture have a mean value of 0.

However, I did choose to add another parameter, brightness. This value is added to add pixels of the image at the same time that those of the texture are. In my testing, I found that lighter images looked less like the original photo and more like a pencil sketch on white paper. Figure 14 shows how I varied the blend and brightness parameters to find a desirable result.

Left: Figure 14

Above: Figure 15

I initially added the colour pencil effect by using the same texture on two channels. I was pleased with the result (figure 15), but then once I tried using different textures on the two channels – as the problem specification requires – the output lost both the pencil and monochrome effect. As figure 16 shows, the image becomes multi-coloured wherever the intensities of the two channels vary significantly. The effect can be reduced a little by using lower noise and blend parameters, however this makes the pencil strokes less pronounced.

Figure 16

The best parameters for the four image / colour mode combinations are listed in the table below.

Input Image	Colour Mode	Blend	Noise	Brightness
face1.jpg	Greyscale	0.5	40	70
face1.jpg	Colour	0.5	40	70
face2.jpg	Greyscale	0.5	50	30
face2.jpg	Colour	0.5	40	70

Figures 17 - 20: The images generated using the parameters above

Computational Complexity

The greatest cost in time complexity comes with the convolution of the mask around the noise image. This has time complexity $O(CRN^2)$, where C and R are the dimensions of the image and N is the height and width of the mask. This is because for every pixel in the image, a multiplication occurs for every cell in the kernel.

Problem 3

Development

I chose to use a bilateral filter to smooth the face. It is ideal because it removes high frequency detail, like imperfections of the skin, whilst leaving the structural parts of the image intact. The spatial factor works in the same way as a Gaussian filter: it places more weight on nearer pixels so smooths out subtle gradients and noise. This alone would leave the image looking too blurry. The intensity factor stops smoothing between pixels of different colour or brightness, so this preserves the sharp edges of the image.

I decided not to build the filter myself because this would take too long to be worth the 5 marks available for this part (stated by the problem specification).

There are three parameters I can change for the bilateral filter. Most sizes of kernel produce similar results, so I chose to use 9. The other two parameters are the standard deviations of the Gaussian functions of intensity difference (σ_1) and proximity (σ_2). I found the sweet spot for σ_1 to be 30 – figures 21, 22 and 23 show σ_1 set to 10, 30 and 50 whilst σ_2 is constant at 100. Set it too low, and too much noise remains, but set it too high, and the face begins to look like a cartoon.

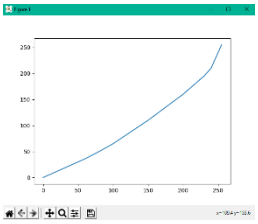
Figures 20 - 23

Below: Figure 24

With σ_1 set to 30, the value of σ_2 does not make a very noticeable difference once it goes over 15 (any lower and too much texture remains). I have it set to 100.

The name of my filter is "Fake Tan Filter". The aim is to darken the pigment of the skin to make the face look more tanned. Before writing any code, I played around with changing various image channels to see what effect each would have. I did this using websites like www.imgonline.com.ua/eng/hue-saturation-luminance.php. I found that increasing the red channel relative to the green and blue did not have the desired effect: the face was now a stronger shade of red, but it didn't look any darker. I then tried adjusting channels in the HSV colour space. I found that reducing the saturation and value by a constant of 30 for every pixel did have the desired effect, although it was a bit too dark (see figure 24).

Figure 25



Now that I knew what factors to change, I created a look-up table (LUT) to map higher values to slightly lower ones using the SciPy Univariate Spline function. Figure 25 is a plot of the result. I then applied this mapping to the HSV image, but the results were different to what the website had suggested. Notably, decreasing both saturation and value made the image look pale and dark. I tested decreasing just one channel but both results were unsatisfactory. I then created the inverse look-up table, so I could increase values in a channel. I tried all combinations of increasing and decreasing both channels, and the results are shown in Figure 26.

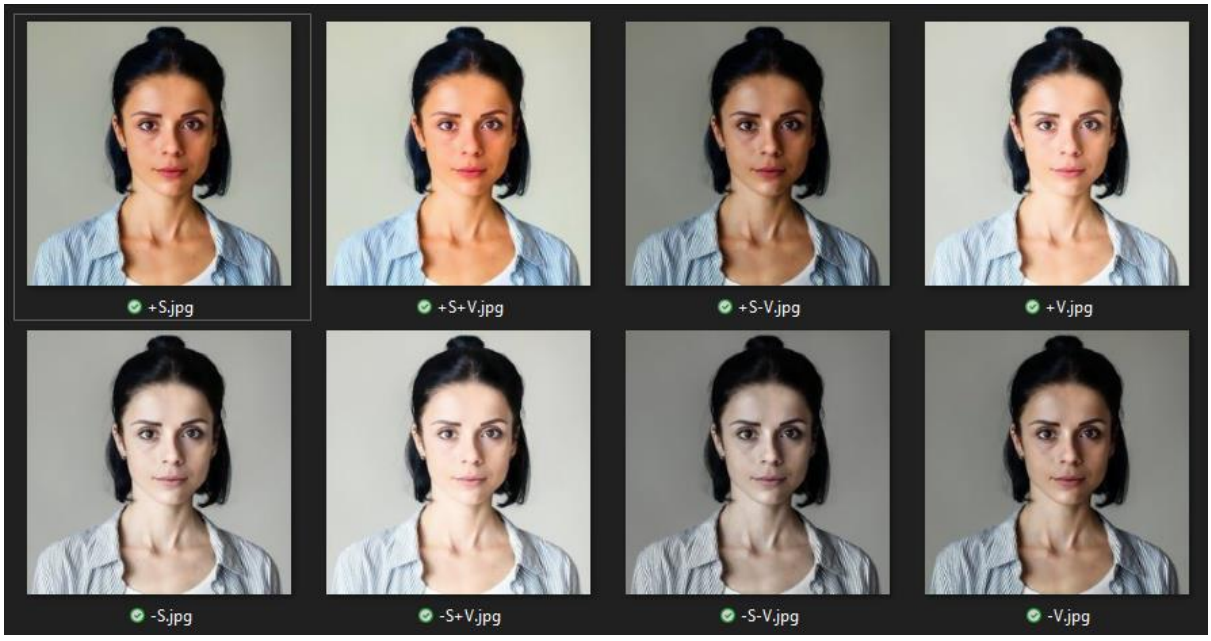


Figure 26: e.g., +S means increased saturation, -V means decreased value

I concluded that increasing the saturation only (see image titled +S.jpg in figure 26) made the skin look more tanned without being too bright or dull so that is what my submitted code does.

Computational Complexity

The bilateral filter convolves a kernel around the image like in Problem 3, so it has the same time complexity of $O(CRN^2)$.

Problem 4

Development

Figure 27 shows the result of my swirl function on image2.jpg, with the parameters set to try and recreate the example in the problem specification as closely as possible. Figure 28 shows the image after a median filter is applied. Figure 29 shows the image after median filtering, followed by the swirl transformation. A lot of definition is lost from the median filter, and the aliasing artifacts were not too severe without pre-filtering, so I would not recommend using this filter. Perhaps bilateral filtering would look better, as the sharp edges of the image would be retained.



I did not have time to successfully implement bilinear interpolation, but I have left my attempt visible in the code.

I did not ultimately have time to implement the inverse transformation, but I did write my code so that it would be easy to implement: the reverse transformation can be achieved by using the negative of the swirl angle parameter.