Commentary

Q8: Image thresholding for each RGB channel

Thresholding each colour channel individually highlights different aspects of an image's structure, as each channel carries distinct intensity variations.
- The red channel accentuates skin tones, making facial features more prominent, likely due to the similarity between red shades and human skin.
- The green channel enhances edge details more effectively than the others, possibly due to its higher weight in luminance calculations.
- The blue channel is the least effective in preserving object clarity, as it tends to have lower luminance and higher noise levels, making features less distinguishable.

Q11: Image thresholding for the Colour Space

There were notable differences in the thresholding results after converting the image to alternative colour spaces
- Color-space conversions generally improved feature separation but also introduced noise.
- HSV and YCbCr were used. The HSV model, which separates intensity from colour information, provided a clearer thresholded result. The YCbCr model, commonly used in image compression, highlighted differences in chromatic details.
- Alternative colour spaces such as Lab* could potentially improve results by isolating brightness from colour, making thresholding more reliable.

Problems faced

Implementing threshold for RGB channel

One of the challenges faced was designing a system that could threshold each channel separately while keeping the code efficient. Initially, I considered creating separate folders for each channel but later optimised the implementation within a single structure.

Face detection

The biggest challenge was implementing face detection effectively. Initially, I used the library recommended in the course but encountered persistent errors. As an alternative, I tried the ml5.js library, which provided additional functionalities but introduced new errors. After debugging, I reverted to the original library and resolved previous issues. Later, while working on the extension, I realised that ml5.js offered more possibilities. At this point, I switched back to ml5.js and, after extensive troubleshooting, successfully integrated it.

Another issue was ensuring that both the face detection and the extension filter appeared in their intended grid positions. I was able to solve it after many trial-and-error with the adjustments to coordinate mapping and display logic.

Target to complete the project

Despite all the challenges faced, I managed to resolve most of them, ensuring my image-processing application ran smoothly. Something that I can improve on is time management, as allocating more structured debugging time would have helped in addressing issues earlier in development.

Extension

For my extension, I implemented a laser-eye filter inspired by social media filters. It uses ml5's faceMesh, to detect the facial landmarks, identify eye positions, and draw animated laser beams extending from the eyes.
This extension is unique because it dynamically tracks facial movements in real time, making it more interactive than a static filter. Future improvements could involve adding color variations, beam width adjustments, or interactive elements where laser intensity changes based on facial expressions.