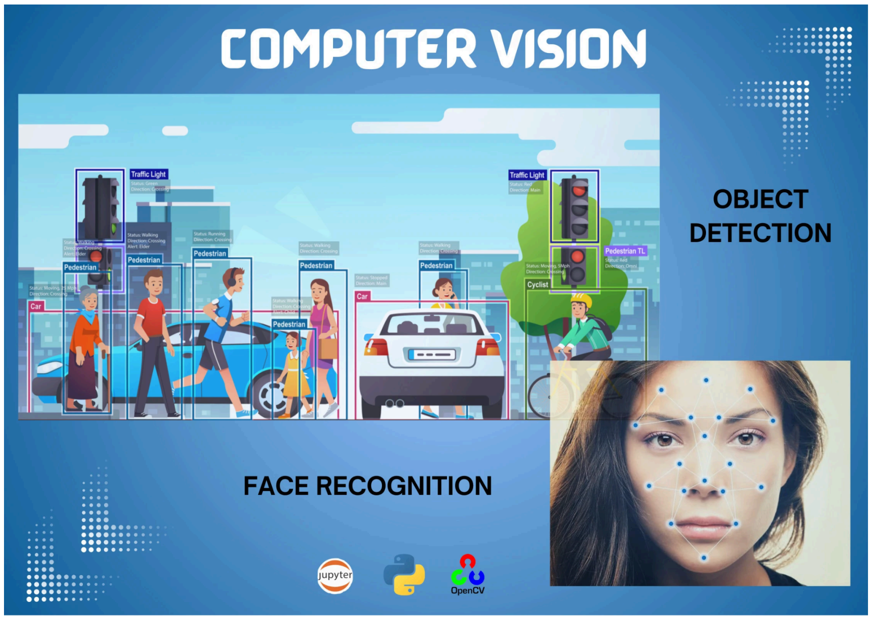
Report for SoC 2024

[Project ID: 183]
Vision Venture: Exploring the world of computer vision



Mentee: Sahil Ravindra Patil (23B0005)

Mentors: Mugdha Bilotia, Anushka Bhoir, and Mahit Gadhiwala

This project was a rewarding experience focused on Computer Vision (CV), with the ultimate goal of achieving motion magnification in images and videos. It was divided into two phases.

In the initial phase, we spent the first few weeks learning Python and key libraries like Numpy and Matplotlib. We then delved into signal processing, covering essential concepts such as the Fourier transform, convolution, and filters at a high level. By the third week, we had begun exploring image processing fundamentals, completing assignments to reinforce our understanding on topics such as image filtering via Per-Pixel Transformations and Correlations, image blurring, sharpening, and edge detection.

The second phase commenced in the fourth week, where we transitioned to learning CV2, which became crucial for our final project. Next in week 5, we studied the theory behind spatial linear filters and multi-scale pyramids, further enriching our understanding of image processing techniques.

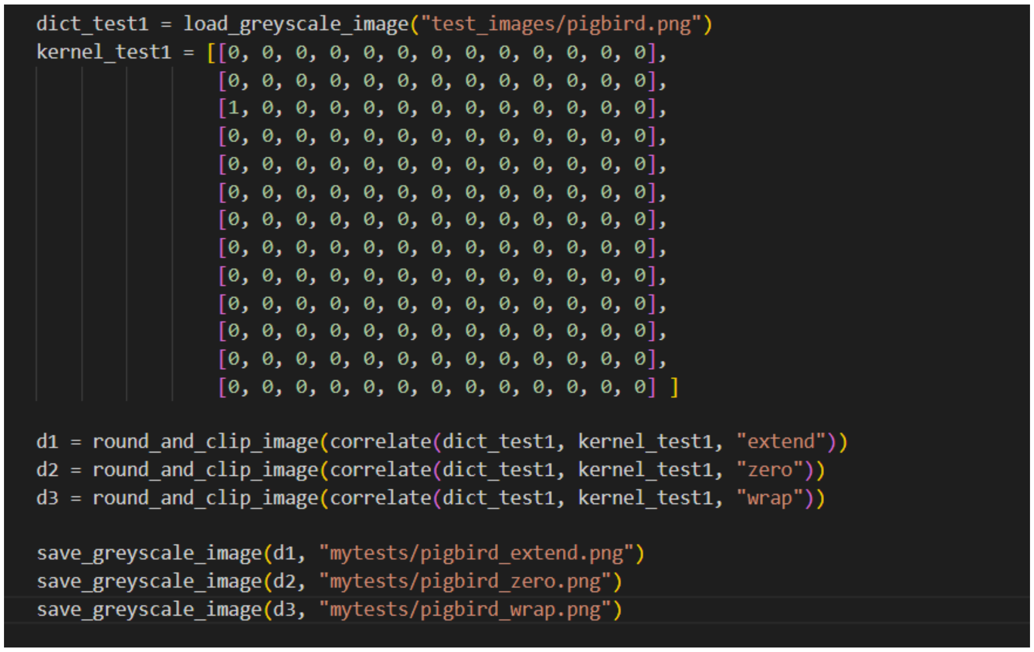
The project culminated in the final assignment, where we applied all the knowledge and skills acquired throughout the course to implement motion magnification of images and videos. This hands-on experience was both educational and enjoyable, providing a comprehensive understanding of the various aspects of Computer Vision.

# Specific reports/analysis on every assignment:

Assignment 1) Was very easy and straightforward.

Assignment 2)

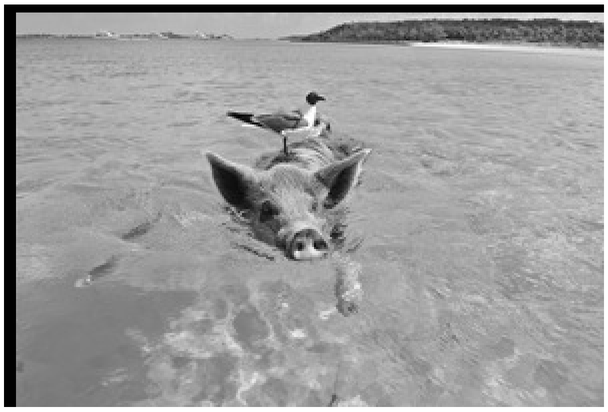
a) Correlation and round_and_clip :

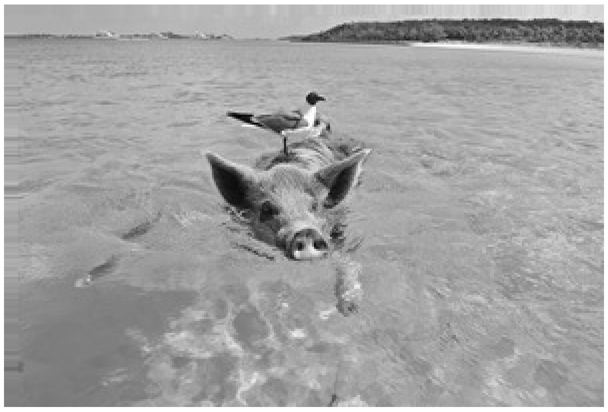


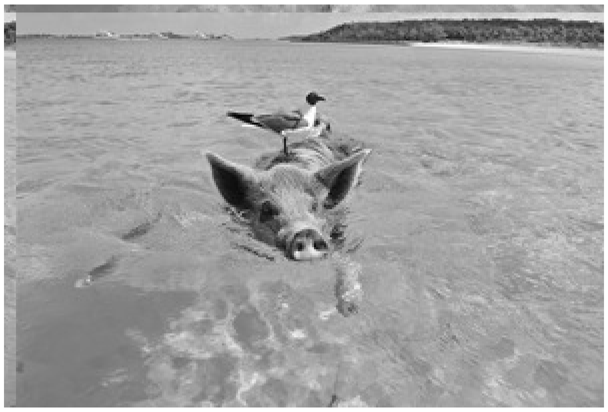
```
dict_test1 = load_greyscale_image("test_images/pigbird.png")
kernel_test1 = [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] ]

d1 = round_and_clip_image(correlate(dict_test1, kernel_test1, "extend"))
d2 = round_and_clip_image(correlate(dict_test1, kernel_test1, "zero"))
d3 = round_and_clip_image(correlate(dict_test1, kernel_test1, "wrap"))

save_greyscale_image(d1, "mytests/pigbird_extend.png")
save_greyscale_image(d2, "mytests/pigbird_zero.png")
save_greyscale_image(d3, "mytests/pigbird_wrap.png")
```


pigbird_zero.png

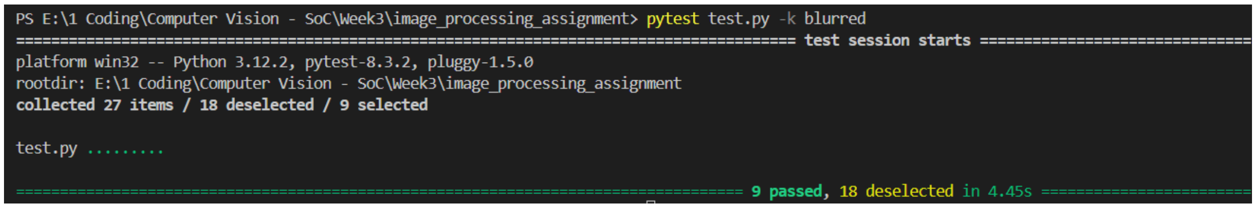

pigbird_extend.png


pigbird_wrap.png

```
How do the different edge-effect behaviors manifest in the output?
```

→ **Zero**: This treats out-of-bound pixels as having a value of 0.
Extend: This replicates the edge pixel values for out-of-bound areas.
Wrap: This wraps around the image, using the opposite edge pixels for out-of-bound areas.

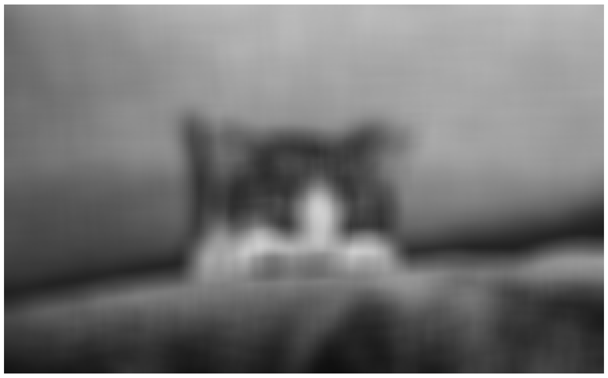
b) Blurring :

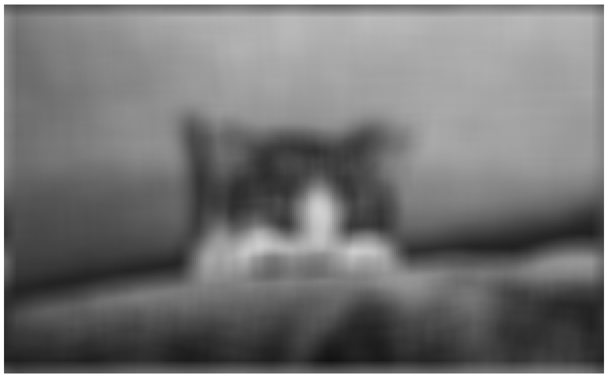


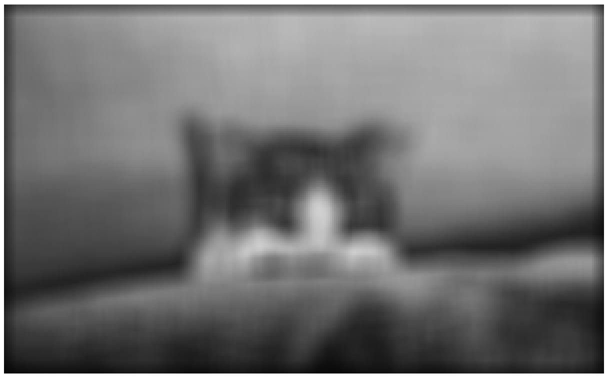
```
PS E:\1 Coding\Computer Vision - SoC\Week3\image_processing_assignment> pytest test.py -k blurred
================================================================================= test session starts ================================
platform win32 -- Python 3.12.2, pytest-8.3.2, pluggy-1.5.0
rootdir: E:\1 Coding\Computer Vision - SoC\Week3\image_processing_assignment
collected 27 items / 18 deselected / 9 selected

test.py .........

=========================================================================== 9 passed, 18 deselected in 4.45s ========================
```


cat_blurred_extend.png


cat_blurred_wrap.png

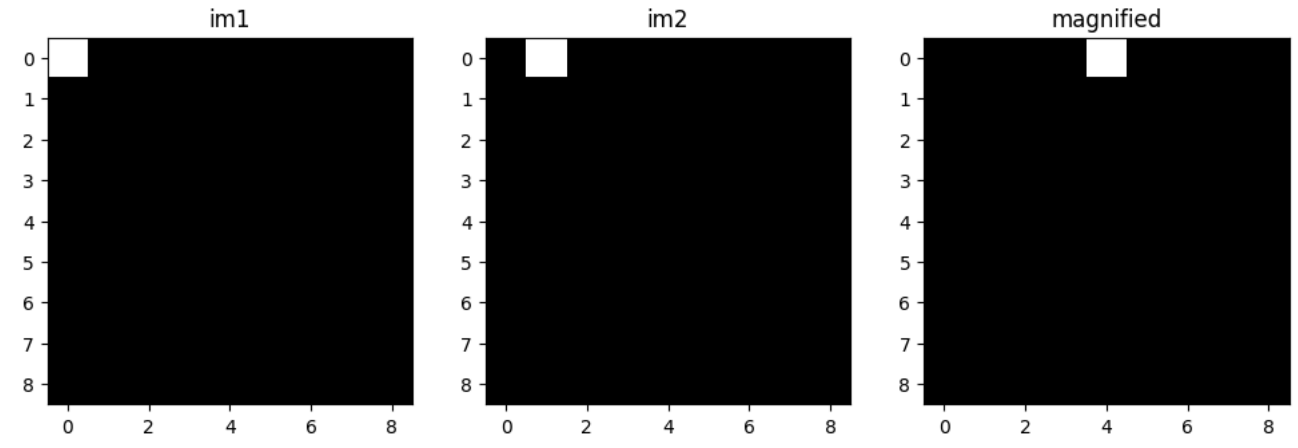

cat_blurred_zero.png

```
Look at those images; why do they look the way they do?
```

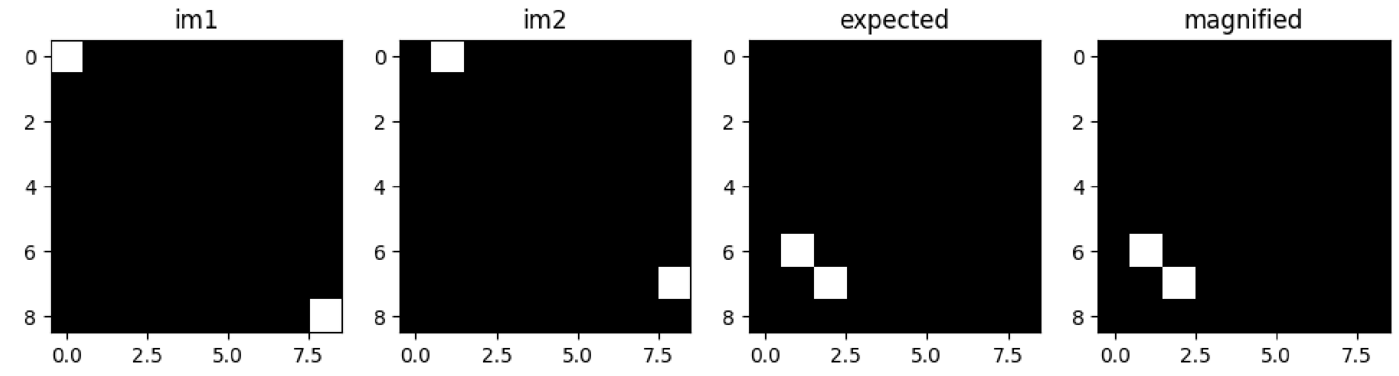
→

Assignment 3)

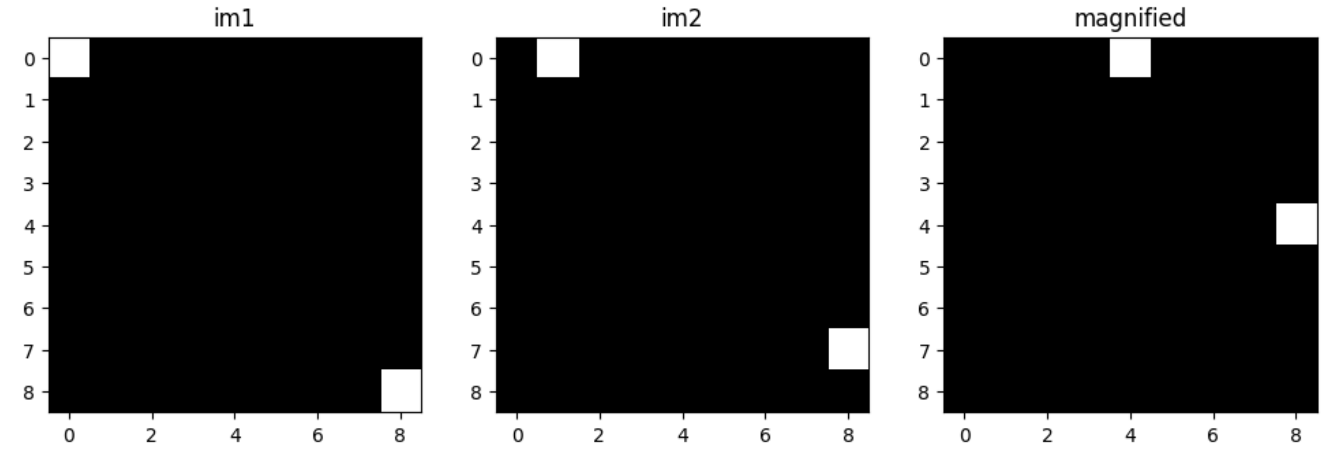
Problem 1a :



Problem 1b :



Problem 1c :



Problem 1d :

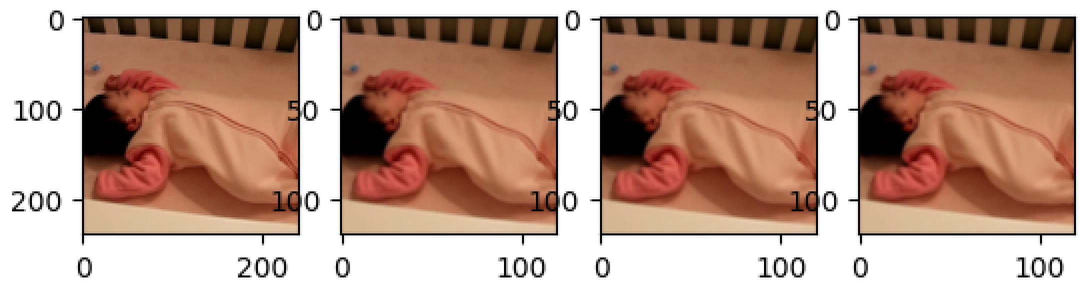
Original :
https://drive.google.com/file/d/1Ip4Dn-cQSeauNF-mKIKCWAoW782iAwiu/view?usp=sharing

Magnified :
https://drive.google.com/file/d/1NEeFWqMURZtG-Mk2Pp9xw_s_Y2KV2Ox9/view?usp=sharing

Problem 2: Eulerian Motion Magnification

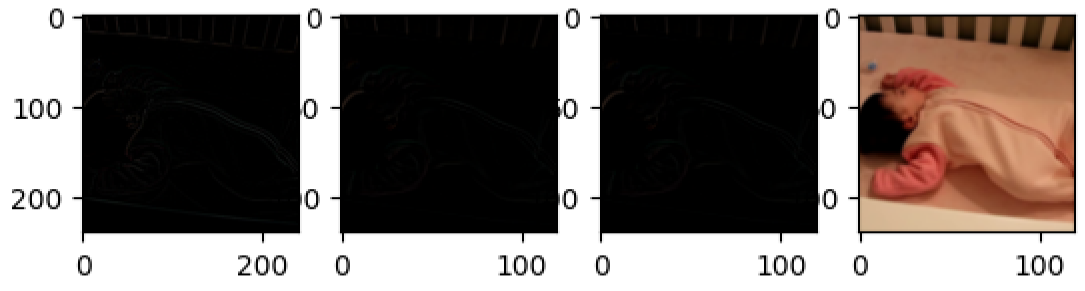
Original video of the baby (baby.mp4)
https://drive.google.com/file/d/1bkJLYtbZ9uQa5UK-9RwAj73FwJ2XHmGo/view?usp=sharing

Problem 2a :



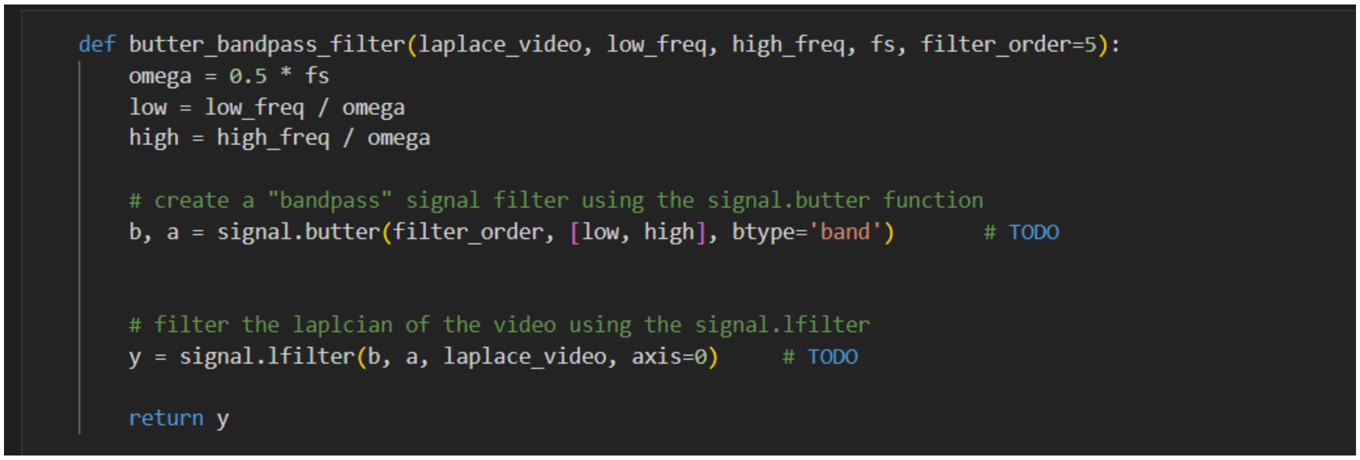
gaussian_pyramid.png

Problem 2b :



laplacian_pyramid.png

Problem 2c :



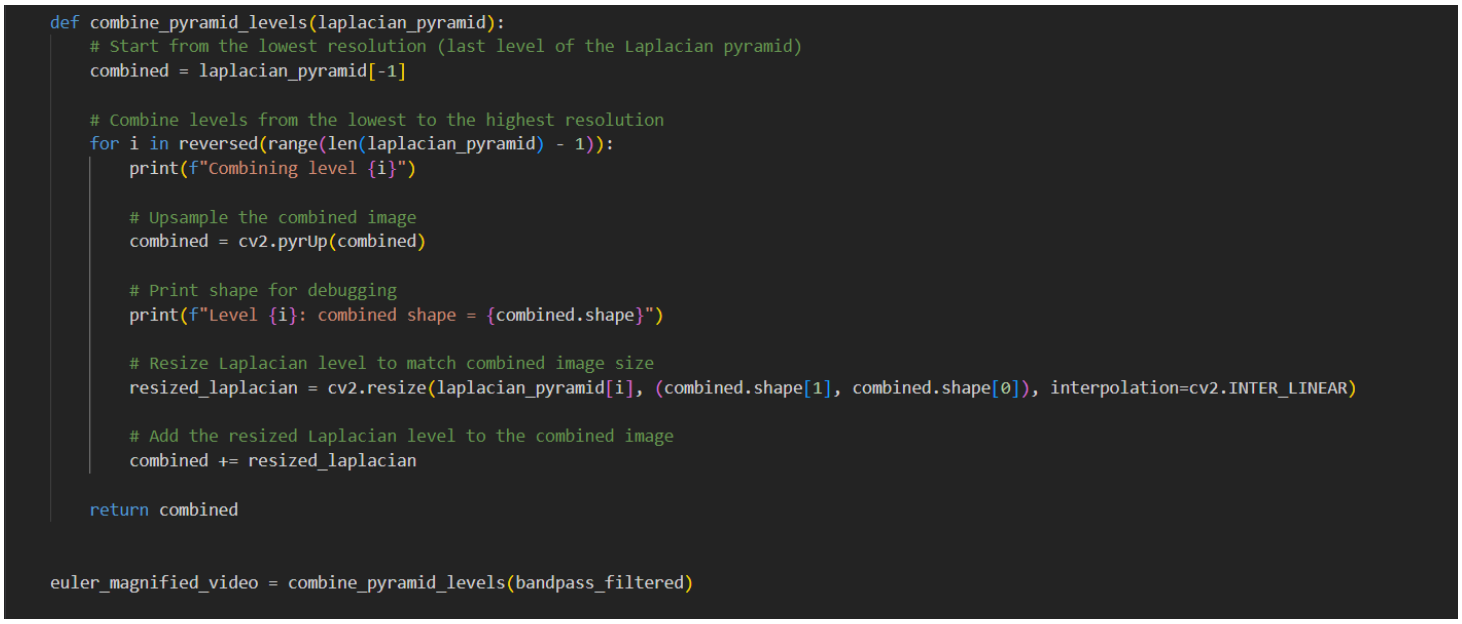
```python
def butter_bandpass_filter(laplace_video, low_freq, high_freq, fs, filter_order=5):
    omega = 0.5 * fs
    low = low_freq / omega
    high = high_freq / omega

    # create a "bandpass" signal filter using the signal.butter function
    b, a = signal.butter(filter_order, [low, high], btype='band')       # TODO


    # filter the laplcian of the video using the signal.lfilter
    y = signal.lfilter(b, a, laplace_video, axis=0)     # TODO

    return y
```

Problem 2d :



```python
def combine_pyramid_levels(laplacian_pyramid):
    # Start from the lowest resolution (last level of the Laplacian pyramid)
    combined = laplacian_pyramid[-1]

    # Combine levels from the lowest to the highest resolution
    for i in reversed(range(len(laplacian_pyramid) - 1)):
        print(f"Combining level {i}")

        # Upsample the combined image
        combined = cv2.pyrUp(combined)

        # Print shape for debugging
        print(f"Level {i}: combined shape = {combined.shape}")

        # Resize Laplacian level to match combined image size
        resized_laplacian = cv2.resize(laplacian_pyramid[i], (combined.shape[1], combined.shape[0]), interpolation=cv2.INTER_LINEAR)

        # Add the resized Laplacian level to the combined image
        combined += resized_laplacian

    return combined


euler_magnified_video = combine_pyramid_levels(bandpass_filtered)
```

End of Report