Name: John Stephen Gutam
Email: jgutam@gmu.edu

Part-1: Corner detector

1) For fixed parameter values, on running the detector on house1.jpg.

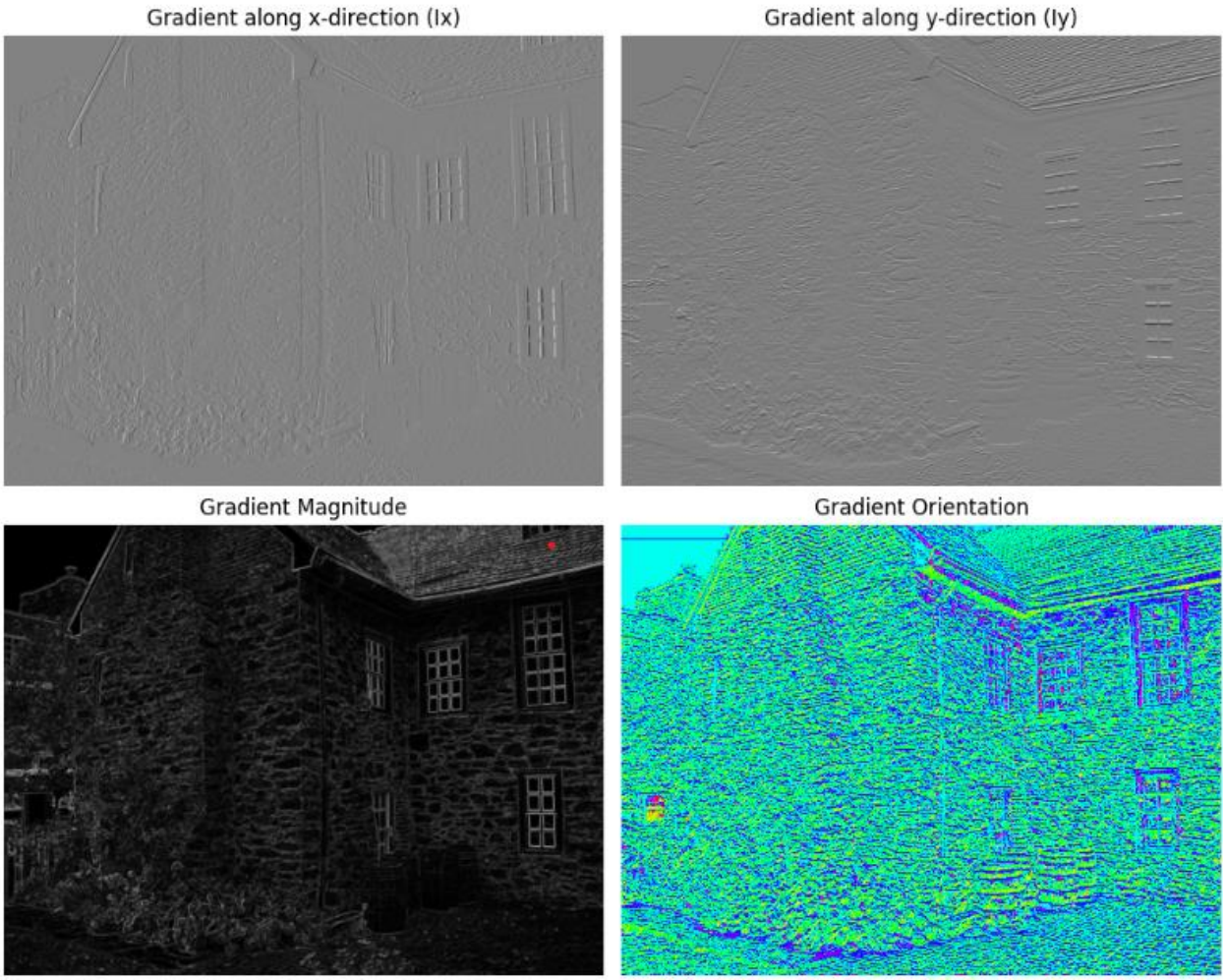
On running the detector on house1-rotated.jpg

If we rotate the input image, the detected corner positions rotate by the same amount. We can observe the same from the above picture. The orientation of the corner detection also changed as we rotated the image.

2) On scaling down the house1-2down.jpg we observe that the red asterick count has been reduced as the image resolution is reduce to the half of its original image.

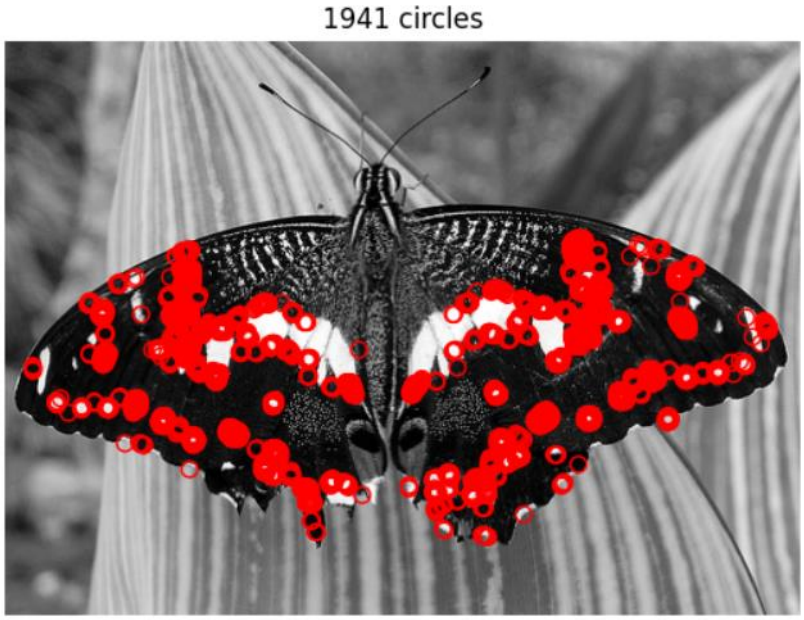
When the same image is furthermore downsized to its half of the previous image, the corner detection which is a red asterisk has been further reduced as shown in the below picture.

3) Below are the derivatives Ix and Iy, gradient magnitude, and gradient orientation of the image house1.jpg.

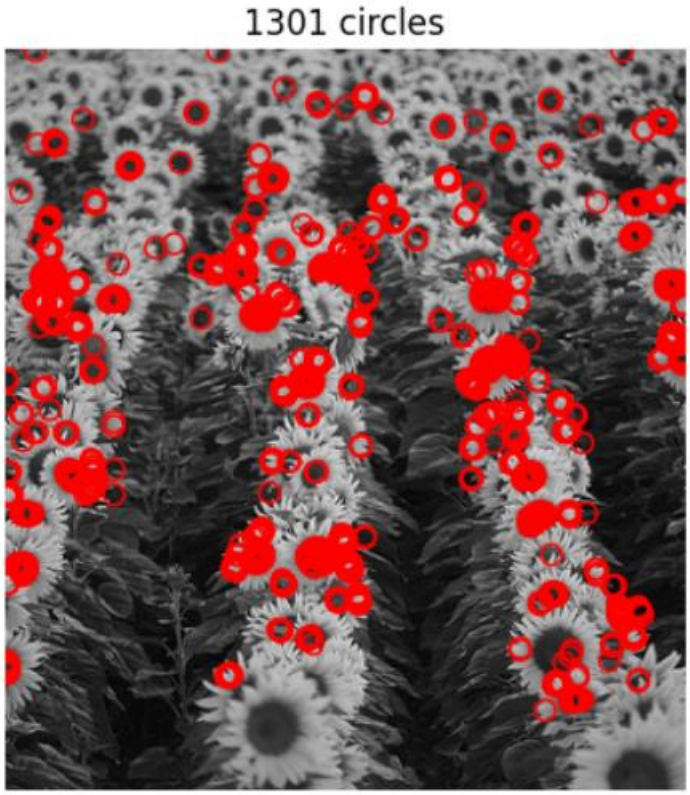


Part-2: Scale-space blob detection

1) The output of the circle detector on all the images (four provided and four of your own choice), together with running times for both the "efficient" and the "inefficient" implementation.

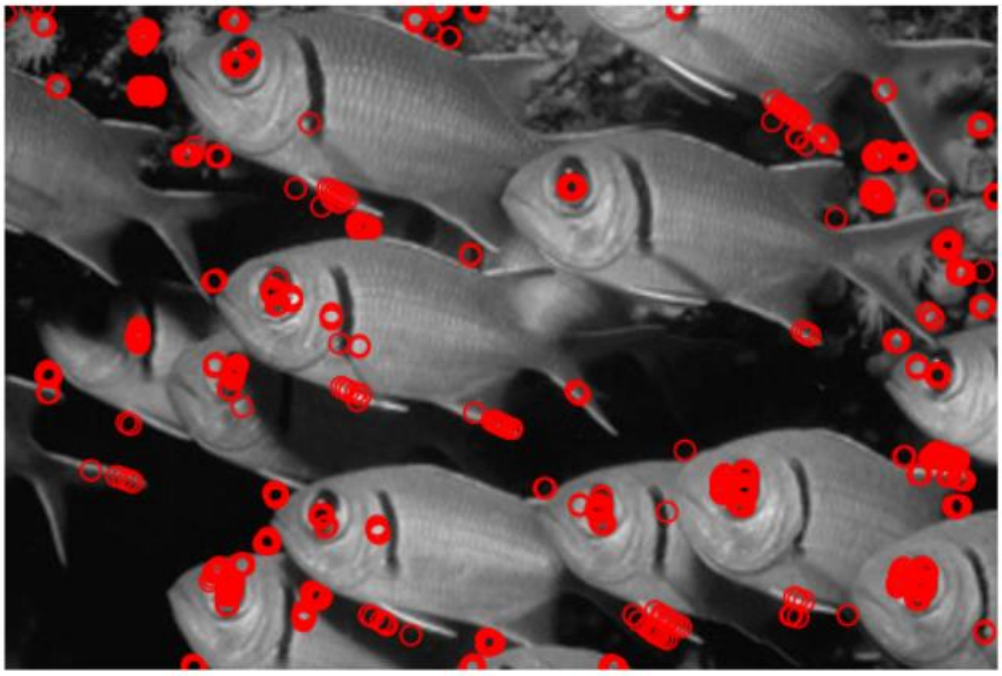


Threshold = 10



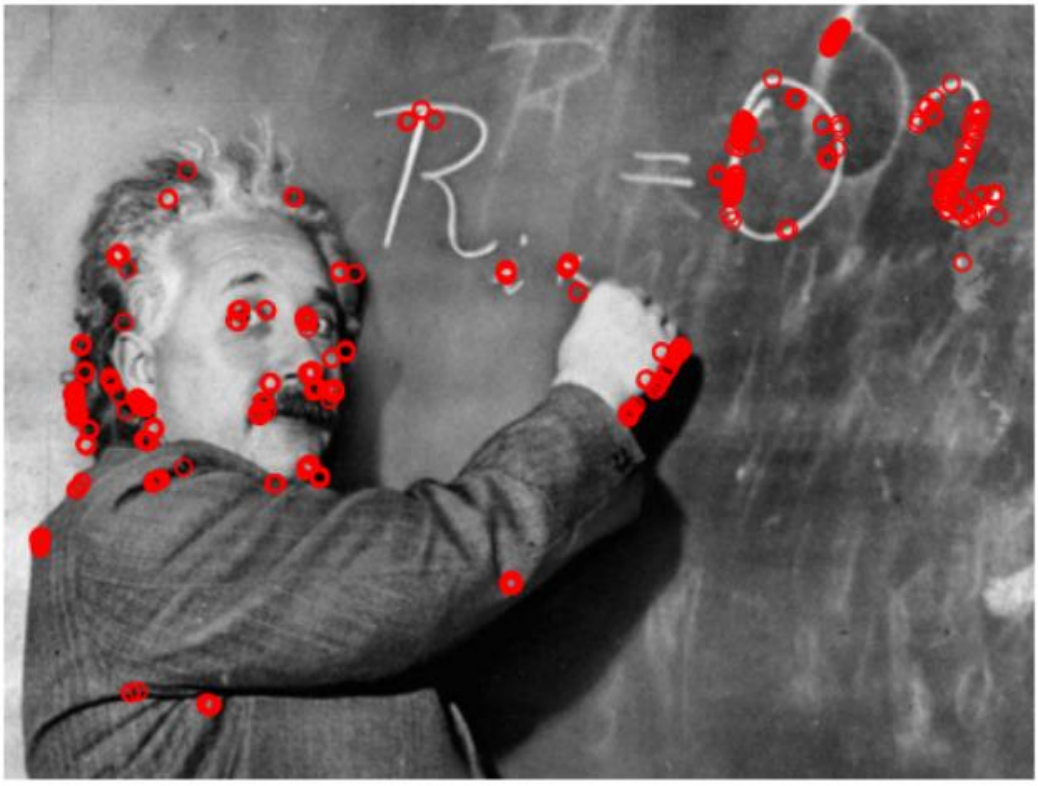
Threshold = 5

817 circles



Threshold 10

496 circles



7914 circles

4328 circles

6722 circles

15872 circles

2) An explanation of any "interesting" implementation choices that you made.

For displaying the blobs, I made use of the given code incorporated into my code by using a few lines of code.

```
def display_blobs(image, blobs):
    """
    # Display resulting circles at their characteristic scales
    image_with_blobs = image.copy()
    for blob in blobs:
        x, y, scale = blob
        radius = int(scale)
        cv2.circle(image_with_blobs, (x, y), radius, (0, 255, 0), 2)
    plt.imshow(image_with_blobs, cmap='gray')
    plt.axis('off')
    plt.show()
    """
    cx = [int(blob[0]) for blob in blobs]
    cy = [int(blob[1]) for blob in blobs]
    rad = [int(blob[2]) for blob in blobs]
    show_all_circles(image, cx, cy, rad, color='r')

def show_all_circles(image, cx, cy, rad, color='r'):
    """
    image: numpy array, representing the grayscsale image
    cx, cy: numpy arrays or lists, centers of the detected blobs
    rad: numpy array or list, radius of the detected blobs
    """
    import matplotlib.pyplot as plt
    from matplotlib.patches import Circle
```

3) An explanation of parameter values you have tried and which ones you found optimal.

The parameter threshold value I tried is 5, 10, 15, 20, etc. Out of which the lower values give me a good plot of finding the features in the image by plotting around the sudden changes in the intensity. Out of these, I found threshold value 10 best suits most of the pictures to give a nice plot.

4) Discussion and results of any extensions or bonus features you have implemented.

In the implemented Laplacian blob detector, we modified the traditional approach by incorporating down sampling of the image instead of increasing the filter size. This change aimed at efficiency while maintaining accurate blob detection across different scales.
As an extension, we could incorporate adaptive down sampling strategies based on image content or blob density. This would dynamically adjust the down sampling factor based on the local image characteristics, potentially improving efficiency further.