ICS 483: Advanced Computer Vision (Fall 2019)

Homework Assignment #3

Gaussian image pyramid and corner detection

(Due: 10:30 AM, Monday, October 14)

Instructions

- Download hw3_images.zip for images used in this assignment.
- Use Python 3.x for all homework assignments in this class.
- For image read/write and basic image manipulations you can use any of the packages/libraries such as Scikit-image, SciPy, PIL (Python image library), and OpenCV, etc.
- Make a single notebook (.ipynb) file. Include answers to questions and any additional explanations/reasoning in the notebook file.
    - In the code, **make sure to type in your name**, clearly specify which question is addressed in each part of the code, and put any necessary comments.
    - To help the grader (and also yourself), please document your code properly.
- Submission: Create "Homework-3" folder in your drop box and upload all the files in that folder.

  Deliverables:

    - Your Jupyter notebook file.
    - Resulting images:
        - butterfly _pyramid_sf2.jpg
        - butterfly _pyramid_sf3_sigma5.jpg
        - butterfly_ pyramid_edges.jpg
        - butterfly_ pyramid_edge0.jpg
        - butterfly_ pyramid_edge1.jpg
        - butterfly_ pyramid_edge2.jpg
        - butterfly_ pyramid_edge3.jpg
        - butterfly_ pyramid_edge4.jpg
        - butterfly_corners_resp1_down2.jpg
        - butterfly_corners_resp1_down4.jpg
        - butterfly_corners_resp1_mindist5.jpg
        - butterfly_corners_resp1_mindist15.jpg
        - butterfly_corners_resp1_rot.jpg
        - butterfly_corners_resp1_tf0.1.jpg
        - butterfly_corners_resp1_tf0.3.jpg
        - butterfly_corners_resp2_down2.jpg
        - butterfly_corners_resp2_down4.jpg
        - butterfly_corners_resp2_mindist5.jpg

- butterfly_corners_resp2_mindist15.jpg
- butterfly_corners_resp2_rot.jpg
- butterfly_corners_resp2_tf0.1.jpg
- butterfly_corners_resp2_tf0.3.jpg

- If necessary, you can provide a README file describing any special instructions that need to be noted for running your code including software or system set-up.

1. (50 points) **Gaussian image pyramids**: An image can be decomposed into multiple scales to extract features/structures of interest at multiple scales and to model images for enhancement, blending, analysis, and synthesis. An example of filter-based multi-scale representation is Gaussian pyramid shown below.

Figure 1. A Gaussian pyramid

a) A Gaussian pyramid is produced by repeating the following two steps:

(1) Apply a Gaussian filter to smooth the image at current scale. This step removes high frequency component of the image.

(2) Down-sample the smoothed image.

Write your own function to construct a Gaussian pyramid,

```
scaled_images = my_gaussian_pyramid(image, scale_factor,
                                    n_scales, sigma)
```

The function takes a grayscale image, downscale factor, number of scale levels, and the standard deviation 'sigma' for Gaussian filter. Set the defaults as follows.

- `scale_factor = 2`
- `n_scales = 5`
- `sigma = 2 * scale_factor / 6.0`

The default value of sigma for Gaussian filter corresponds to a filter mask twice the size of the scale factor that covers more than 99% of the Gaussian distribution. The function returns a three-dimensional array representing your Gaussian pyramid.

Notes for implementation:

- In the function, you should compute the maximum number of scales (max_scales) given the image size and the downscale factor, and set the number of scales of the pyramid as min(max_scales, n_scals). That is, the number of scales in the output pyramid becomes max_scales if n_scales > max_scales.
- You can use gaussian_filter function in the SciPy library for smoothing. However, you should implement your own version of down-sampling with given scaling factor.

b) Apply your function to "butterfly.png" with the following parameter settings:

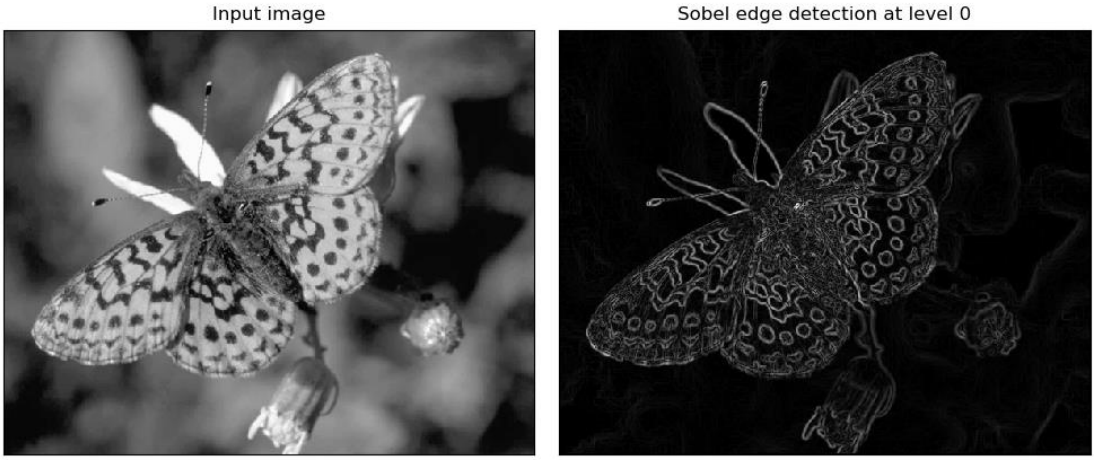
(1) scale_factor = 2, n_scales = 5 (default sigma value is used)

(2) scale_factor = 3, n_scales = 10, sigma = 5

For both case, make a single composite image containing all the scales of the Gaussian pyramid as shown in Figure 1. Save the resulting image as "butterfly _pyramid_sf2.jpg" for (1) and "butterfly _pyramid_sf3_sigma5.jpg" for (2).

c) Using `sobel` function in scikit-image library, convolve each image in the Gaussian pyramid obtained in (b)(1) with Sobel filter. Make a single composite image as in (b) showing the resulting magnitude images at all scales. Save the resulting image as "butterfly _pyramid_edges.jpg"

d) Resized the resulting gradient magnitude images to match the size of input image. (You may want to use the `resize` function in scikit-image library.) For each scale, display the resized gradient magnitude image along with the input image and put appropriate title for each image in the display. Below shows an example plot for the image at scale 0 (or level 0).



- Describe your observation on the detected feature/structure in relation with the scales in the image pyramid.
- Save the displayed image as "butterfly_pyramid_edge{*n*}.jpg", where *n* goes from 0 to 4.

2. (50 points) **Corner detection**: Harris corner detection is a simple algorithm for locating interest points where the surrounding neighborhood contains edges in more than one direction. The outline of the Harris corner detection algorithm is as follows:

   - Step 1: Compute derivatives of the input image in *x*- and *y*-direction.
   - Step 2: Construct the second moment matrix M in a Gaussian window around each pixel.
   - Step 3: Compute the corner response function.
   - Step 4: Threshold the corner response value computed above.
   - Step 5: Find local maxima of response function (non-maximum suppression)
   - Step 6: Display resulting corners.

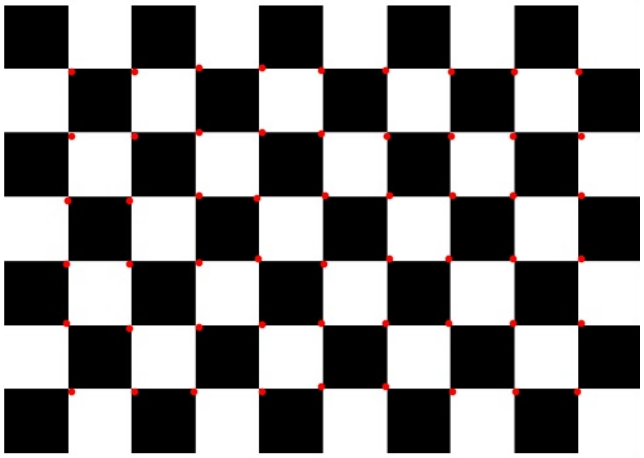
   a) Implement your own version of Harris corner detector. You can verify your implementation using "checkerboard.jpg".

   Notes for implementation:

   - Your code should take thresholding factor used in Step 4 and the minimum distance between corners used for non-maximum suppression in Step 5 as parameters. (Do not hardcode these values.) Set the default values for thresholding factor and minimum distance as 0.1 and 10, respectively.
   - You can use built-in functions for convolving with Gaussians. However, your code cannot call on any existing functions for other processes such as thresholding and finding local maxima of corner responses.
   - Set $\sigma = 1$ for the Gaussians used in Step 1 and Step 2.
   - In Step 3, implement two options for computing corner response:

     (1) $R = \det(M) - \alpha * trace(M)^2$. (Set $\alpha = 0.05$ as default value.)

     (2) $R = \det(M) \,/\, (trace(M) + 10^{-6})$
   - In Step 4, set threshold = thresholding_factor * max_corner_response.
   - In Step 6, show resulting corners as red dots on top of the input image as shown below.



b) Apply your implementation of Harris corner detection to “butterfly.png” for each of the two corner response function in Step 3.

- Try 0.1*(max_corner_response) and 0.3*(max_corner_response) for thresholding. (Use default value for minimum distance in Step 5.)
  - Save the results as “butterfly_corners_resp{*m*}_tf{*n*}.jpg”, where *m* = 1, 2 and *n* = 0.1, 0.3.
  - How do the detection results change? Describe your observation.
- Try minimum distance 5 and 15 for finding local maxima. (Use default value for thresholding factor in Step 4.)
  - Save the results as “butterfly_corners_resp{*m*}_mindist{*n*}.jpg”, where *m* = 1, 2 and *n* = 5, 15.
  - How do the detection results change? Describe your observation.

c) Run your code on “butterfly_rotated.png” for both corner response functions using default parameter values.

- Save the results as “butterfly_corner_resp{*m*}_rot.jpg”, where *m* = 1, 2.
- Compare the results with butterfly_corner_resp{*m*}_tf0.1.jpg. Do the detected corner positions rotate by the same amount? Which of the two corner response functions looks more robust to image rotation?

d) We can test whether all the detected corner positions are scaled accordingly when we scale down the input image. “butterfly_down2.png” is half the size of “butterfly.png” and “butterfly_down4.png” is the half the size of “butterfly_down2.png”. Run your code on “butterfly_down2.png” and “butterfly_down4.png” for both corner response functions using default parameter values.

- Save the results as “butterfly_corner_resp{*m*}_rot.jpg”, where *m* = 1, 2.
- Compare the results with butterfly_corner_resp{*m*}_tf0.1.jpg. Are the detected corner positions scaled down accordingly? Which of the two corner response functions looks more robust to scale change?