# Image Processing: Assignment #5

Submission date: 09\02\2025, 07:59.

Please submit the assignment via Moodle.

For questions regarding the assignment please contact:

Alon Papini (imageprocessinghaifau@gmail.com).

## Assignment Instructions:

- Submissions in ***pairs***.
- The code must be written in Python 3.11 or higher.
- The code must be reasonably ***documented***.
- Please submit one single ***.zip*** whose name should be using this format: ID1_ID2_HW5.zip. It should contain ***only***:
  - 'q1' folder, and inside it will be:
    - The three images you were given.
    - Two output images with all matches found.
    - Your python script, 'matchFaces.py'.
  - 'q2' folder, and inside it will be:
    - The two images you were given.
    - The output blended image.
    - Your python script, 'blendFruits.py'
  - 'bonus' folder (*ONLY IF DONE*), and inside it will be:
    - The five images you were given.
    - Your output images with your detections.
    - Your python script, 'heartTransform.py'
  - The PDF file where your written answers will be (including all images you were asked to create or comment on).

## Assignment Topics:

- Multiband resolution spline
- Template matching
- Edge detections and Hough transform

## Good Luck 😊

## Problem 1 – Template matching (50 points):

We want to detect all the faces within a given image and we are going to do that for two different images, with the same face template. To get started on that, do the following in the python script file ‘matchFaces.py’:

a) Implement the function 'scale_down(image, resize_ratio)'.

   Given an image, use a Fourier transform to scale down the image (optimal interpolation, remember?). Leave ratio*size from the original image(Also, consider possibly blurring the image before scaling it down).

   Tips:

   - Assume the ratio is between 0 and 1.
   - Try to implement at first with ratio = 0.5 to get a feel for it.

b) Implement the function 'scale_up(image, resize_ratio)'.

   Given an image, use a Fourier transform to scale up the image. The output should be ratio*size from the original image. You may assume the ratio is at least 1.

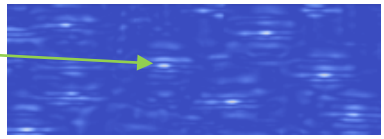
c) Implement the function 'ncc_2d(image, pattern)'.

   Given an image and a pattern image, returns the NCC image (as we’ve seen in the lecture about image matching).

   - Use np.lib.stride_tricks.sliding_window_view(image, pattern.shape) to create all the possible windows from the image, with pattern's size.
   - If the denominator is 0, that means that one of the patches has variance zero => It is constant. In that case, we want to set their cross-correlation to be 0 because it can't really tell us anything.

d) The function 'display' gets an image and a pattern, and displays both, along with their NCC heatmap.

   Just displaying the original image and pattern won’t mean much to you, but with this function you should be able to choose whether to scale the image, the pattern or both, and by how much, such that they will match.

Example for a match: 

We want to filter good matches by their NCC. For example, if NCC[y,x] is 0.2, it is a weak value. Therefore, we will be using thresholding to obtain the appropriate pixels. The threshold value is up to your choice.

e) Using the filtered matches you found, call the function 'draw_matches' with the original image.

The result will be the same image but with red rectangles over the recognized faces.

…But wait, it's not so straightforward. If you chose to scale the image itself, you have to adjust the matched locations relative to the original image, as you saw in the lectures (Scaling things up and down needs to be accounted for, after all).

For this question you'll need to fully implement everything stated above and actually use it to find as many face matches as you can in both 'students.jpg' and 'thecrew.jpg'.

Please save the output images after 'draw_matches' does it work on both images in the same folder as the 'q2' folder, and also attach them in the PDF file along with (you guessed it) explanations of your work throughout this question and include any additional notes and screenshots you think are relevant.

## Problem 2 – Multiband blending (50 points)

Let's go back a bit - do you remember the blending of an apple and an orange from the lecture about multi-scale representation? Great! Now we are going to implement it! It is recommended to go over this section again from lectures before approaching this question. In 'blendFruits.py':

a) Implement the function 'get_laplacian_pyramid'. It gets an image and returns a Laplacian pyramid with the specified levels number. Each additional level is half-size per-axis compared to the previous one.

b) Implement the function 'restore_from_pyramid'. It gets a Laplacian pyramid and returns the image (Remember "collapsing" a Laplacian Pyramid?).

   Use validate_operation(img) to make sure your implementations are correct. Getting less than 1 MSE is good here.

   Now for the tricky part…

c) Implement the function 'blend_pyramids'. To make life just a bit easier for you (but not TOO easy, as this is still a 50-point question), you'll be given an implementation guide here:

   For each level (1 -> total_levels) in the pyramids:

   - Define a mask in the size of the current pyramid.
   - Initialize the mask's columns (remember, it's basically a 2D matrix) from the first one up to (0.5 * width - curr_level) to 1.0 (some advanced numpy indexing action will be required here). We're doing this to properly scale the blending according to the pyramid level.
   - For each column i in the range of (0.5 * width + curr_level), set the value to 0.9 - 0.9 * i / (2 * curr_level). This is gradual blending part.
   - Finally, the blended pyramid level for curr_level is given by:

     orange * mask + apple * (1 - mask)

     Cross dissolve, remember? This formula satisfies the cross-dissolve properly, as it starts from 0.9 and ends in 0. The higher the pyramid level is (the lower frequencies) the wider the dissolving is.

d) The new image is the blending between ‘orange.jpg’ and ‘apple.jpg’. Create a Laplacian Pyramid for each of the two images, blend those two pyramids per-level and then restore the blended image from the result pyramid (do all of this using the functions you’ve implemented).

   Please save the output blended image to the ‘q2’ folder, and also attach it along with explanations about your work (including relevant notes and screenshots) in the PDF file you’ll submit.

   Challenge: Can you get a better result than what the implementation guide for ‘blend_pyramids’ gives? You probably can, as there are MANY different ways to implement the blending function. If you think you can, save your improved result in the ‘q2’ folder and also attach it to the PDF as well, while calling some attention to it. You never know, the teaching assistant *might* feel inclined to give you a small bonus… 😉

## Problem 3 – Hough transform (Up to 15 points bonus):

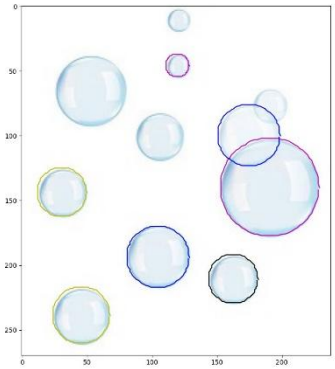
In the lectures you saw an algorithm for line detection and talked about the fact it can be done to circles as well. Before we proceed, recall the algorithm from the lecture:

1. Initialize H[d, θ] = 0
2. For each edge point I[x,y] in the image:
   - For θ = [$\theta_{min}$ to $\theta_{max}$] // some quantization
     i. d = x·cos(θ) - y·sin(θ)
     ii. H[d, θ] += 1
3. Find the value(s) of (d, θ) where H[d, θ] is maximum
4. The detected line in the image is given by d = x·cos(θ) - y·sin(θ)

That same algorithm can be done for every parametric shape. For example, the parametric representation of circle is $(a + r\ cos\ t\ , b + r\ sin\ t\ )$, where (a,b) is the center and r is the radius (Desmos example). The algorithm will be:

1. initialize H[a,b,r] = 0
2. For each edge point I[x,y] in the image
   - For $\theta = [\theta_{min}, \ldots, \theta_{max}]$, $r = [r_{min}, \ldots, r_{max}]$
     i. a = x - r · cos($\theta$), b = y - r · sin($\theta$).
     ii. H[a,b,r] += 1
3. Filter all (a,b,c) in H whose value is beyond some threshold.

Look at the attached image in the bonus folder, 'bubbles_detected.jpg' - it's the result from applying a Hough transform of circles to the 'bubbles.jpg' image:



We will apply the same idea, but for hearts! Here's an example for a heart's parametric equation:

$$(a + 14.5 \cdot r \cdot \sin^3(t), b + r \cdot (0.5 \cdot \cos(4t) + 2 \cdot \cos(3t) + 4 \cdot \cos(2t) - 13 \cdot \cos(t)))$$

Please view the Desmos example here.

Given 'heartTransform.py' to work with, your task is to:

a) Complete the missing implementations in the code.
b) Use an existing canny implementation to find edges in the image.
c) For each image, find the parameters (r_min, r_max, bin_threshold) which give the best results.

You're given 3 images to work on – 'simple.jpg', 'med.jpg' and 'hard.jpg'. For each image you'll be given up to 5 points bonus if you successfully detected and outlined the hearts (in simple it's just the one heart, in med it's two hearts and in hard you have *a lot* of hearts, so you're expected to detect the big, middle one as well as several of the smaller ones surrounding it at least).

For each image, save the output image of your detections on top of the original image (a screenshot of the result plot will also be acceptable, but please – no pictures taken from a phone/camera) to the 'bonus' folder and also attach it, along with relevant parameters, to the PDF file. As usual, please include explanations, notes and relevant screenshots of your work throughout the bonus question.

Remember – this is a difficult bonus question, and considering the difficult level of 'hard.jpg', getting all 15 points is NOT guaranteed!