Assignment 2 – CV1 WS14/15 10/11/2014

Prof. Stefan Roth
Jochen Gast
Stephan Richter

This assignment is due on November 24th, 2014 at 13:00.

Please refer to the previous assignments for general instructions and follow the handin process from there.

UPDATE 1: For subsequent assignments we DO ONLY ACCEPT HANDINS THROUGH MOODLE. The only exception are students who are not affiliated with TU Darmstadt and consequently do not have access to Moodle. For everyone else, in the future you will not receive any points for submissions made via E-Mail.

UPDATE 2: There are still students who do not have a second assignment partner. Please use the Moodle Questions & Answers Forum to get in contact with other students. Otherwise, we will manually assign you into different groups.

UPDATE 3: Since various students complained about that, we have changed the skeleton such that each problem has its own subfolder now. Note, however, that we still use a skeleton with various implementation files, as it allows us to test your code much more efficiently. Also do only include your CODE directory into your submission, if not otherwise specified. We will make use of the same folder structure to run your code.

Problem 1 - Image Pyramids and Image Sharpening (15 points)

Image pyramids are widely used in computer vision. In this problem you will create a small sharpening application by using both Gaussian and Laplacian image pyramids. You will start, however, by writing a few helper functions. Please note that you are not allowed to use the Matlab built-in functions `imresize` and `fspecial` for this problem:

- Function `make_gaussian_filter.m` with two arguments that creates a Gaussian filter with specified size (number of rows and columns) and specified kernel width (standard deviation σ). You can find the formula in the slides. Make sure that the maximum is in the middle of the filter mask (for even and odd filter sizes), and also ensure that the filter is properly normalized (i.e. the filter coefficients should add to 1).
- Function `make_binomial_filter.m` that creates a binomial filter with the specified filter size (number of rows and columns). To construct a binomial filter you initially compute the set of binomial coefficients, e.g. as in Pascal's triangle, and normalize afterwards. For instance, the weights w_k of a 1D binomial filter with $N+1$ weights can be constructed by formula
$$w_k = \widehat{w}_k / \sum_{l=0}^{N} \widehat{w}_l, \quad \widehat{w}_k = \binom{N}{k}, \qquad k = 0, 1, \ldots, N. \tag{1}$$
- Function `downsample2.m` that takes an image and downsamples it by a factor of 2, i.e. resizes both dimensions to half the size. To that end simply discard every other row and column. Note: Do not perform any Gaussian smoothing here; this will be done later.
- Function `upsample2.m` that takes an low resolution image as well as a filter size and upsamples the image by a factor of 2, i.e. resizes each dimension to double the size. In particular, insert one zero row between every low-resolution row and then insert one zero column between every "old" column. Filter the result with a binomial kernel of the given size and finally apply a scale factor of 4. Note: You should make use of your function `make_binomial_filter` and use symmetric boundary conditions for filtering (i.e. the image outside the valid region is filled by mirror-reflecting the image across the borders).

The outline for the sharpening application is given in `problem1.m` and should be completed with the necessary calls. Continue to finish the following tasks by using the functions you wrote above:

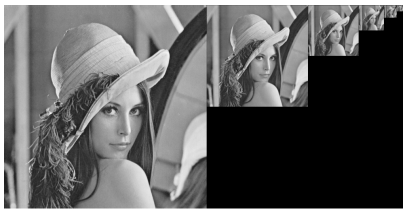

Figure 1: Gaussian pyramid

1. Function `make_gaussianpyramid.m` that builds a multi-level Gaussian pyramid of the image `a2p1.png`. To create a subsequent level of the Gaussian pyramid, filter the image with a Gaussian kernel ($\sigma = 1.5$ and size 5×5) and then downsample by a factor of 2. For filtering steps, apply symmetric boundary conditions and make sure that filtered images have the same size as corresponding inputs. Note: The result of this function should be a cell array containing images with decreasing sizes.

2. Function `display_pyramid.m` that shows image pyramids in a single figure as depicted in Figure 1 . Since we want to use the same function to display both Gaussian and Laplacian image pyramids, you should normalize images of all cells such that they have values $img \in [0, 1]$, respectively.

3. Function `make_laplacianpyramid.m` that, based on the Gaussian pyramid, creates the corresponding Laplacian pyramid. For building the Laplacian pyramid, you should also use the upsampling function from above. What is the difference between the top (coarsest) level of Gaussian and Laplacian pyramids? Please answer that question briefly in `answers_problem1.txt`. Also display the Laplacian pyramid using `display_pyramid.m`

4. In our skeleton we load and sharpen the image `a2p1.png`. To make the sharpening work you have to implement the function `amplify_high_freq2.m` that amplifies the high-frequency components contained in the finest two levels of the Laplacian pyramid by scaling them up by a factor. Afterwards, implement `reconstruct_laplacianpyramid` that reassembles the pyramid back to obtain a sharpened full-resolution image. Try various amplification factors for both levels and choose those that lead to good or interesting results. You may find that the image noise gets amplified if you scale up the coefficients of the finest sub-band too much; try to avoid that. Briefly explain your findings in `answers_problem1.txt`. Finally, display the original image, its reconstruction and their difference in one Figure next to each other.

Problem 2 - PCA for Face Images (15 points)

You will be working with a training database of human face images and build a low-dimensional model of the face appearance using Principal Component Analysis (PCA). The outline is given in `problem2.m` and should be completed with the necessary function calls. Your tasks are:

1. All faces are located in the `data/yale_faces` directory. Implement the function `load_faces.m` that loads all face images into a big data matrix.

2. Implement the PCA of all face images in `compute_pca.m`. Each face image has over 8000 pixels and there are many fewer training images than this. Consequently, you want to use SVD as we discussed in class; in particular you want to use the “economy mode”. (`[U,S,V] = SVD(X,0)` in Matlab). The output arguments of the PCA function contain the PCA basis vectors, their Eigenvalues as well as the the data mean. Make sure that the principal components are properly sorted in decreasing order. Also compute a vector that contains the cumulative variance of the principal components.

3. Show a plot of the cumulative variance of the principal components. Briefly explain in `answers_problem2.txt` how you compute the variance of a single principal component and why the variance can be obtained in that way.

4. Implement the function `compute_ncomponents.m` to compute how many bases account for 80% and 95% of the variance, respectively. Print these numbers on the command window.

5. Display the mean face and the first ten Eigenfaces in a single figure by implementing `display_faces.m`.

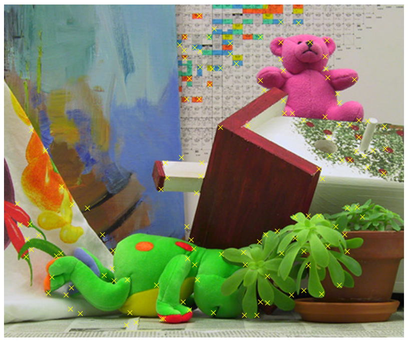

Figure 2: Harris points.

6. Choose a face from your data matrix by implementing `take_face.m`. Project the chosen image onto a low-dimensional subspace using 5/15/50/150 components, respectively. Then implement the function `compute_reconstruction.m` to compute reconstructions from these low-dimensional projections. Finally, implement `display_faces.m` to show the 4 reconstructed faces as well as the original in a single figure.

Problem 3 - Harris Detector (10 points)

The Harris detector identifies corner-like structures by searching for points $p = (x, y)$ for which the structure tensor $\mathbf{C}$ around p has two large eigenvalues. The matrix $\mathbf{C}$ can be computed from the first derivatives at p, spatially weighted by a Gaussian filter kernel $G(\tilde{\sigma})$:

$$\mathbf{C}(\sigma,\tilde{\sigma}) = G(\tilde{\sigma}) * \begin{bmatrix} D_x^2(\sigma) & D_xD_y(\sigma) \\ D_xD_y(\sigma) & D_y^2(\sigma) \end{bmatrix} = \begin{bmatrix} G(\tilde{\sigma}) * D_x^2(\sigma) & G(\tilde{\sigma}) * D_xD_y(\sigma) \\ G(\tilde{\sigma}) * D_xD_y(\sigma) & G(\tilde{\sigma}) * D_y^2(\sigma) \end{bmatrix}, \tag{2}$$

where $*$ is the convolution operator, and $D_x(\sigma), D_y(\sigma)$ denote the partial (horizontal and vertical) derivatives of the image that has been smoothed using a Gaussian smoothing filter with kernel width σ (Note the difference between σ and $\tilde{\sigma}$). The interest points are defined as those points whose Harris function values are larger than a certain threshold t

$$\det(\sigma^2\mathbf{C}) - \alpha \cdot \left(\operatorname{trace}(\sigma^2\mathbf{C})\right)^2 > t. \tag{3}$$

Note that we include an additional scale normalization factor σ^2 so that we can use the same threshold t independently of the value of σ. In practice, the parameters are usually set as: $\tilde{\sigma} = 1.6\sigma, \alpha = 0.06$. For the following tasks please use the Matlab built-in function `fspecial` to generate Gaussian smoothing filters with size 25×25 and use central differences to compute the derivatives. For color images you only need to detect the interest points in the gray-scale space (`rgb2gray` might be useful here). The code outline is given in `problem3.m` which should be completed with the necessary function calls. Your tasks are:

1. Function `load_image.m` that is used to load both the gray-scale and color version of `a2p3.png`.

2. Function `compute_tensor.m` to obtain the structure tensor $\mathbf{C}$ with $\sigma = 2.4$ as computed above. Use replicate boundary conditions for any filtering involved.

3. Function `compute_harris.m` that computes Harris function values given by the left-hand side of (3). Also display these values as an image.

4. Function `nonmaxsupp.m` that applies non-maximum suppression to the Harris function values in order to extract local maxima, *i.e.*, points for which Harris function values are the largest within their surrounding 5×5 windows, respectively. Allow multiple equal maxima in one window and do not pad at the image boundaries. After that find all local maxima with a Harris function value that is larger than the threshold $t = 10^{-6}$. Mark these interest points with "$\times$" in the original image. To achieve this please use `plot(·,·,'x')` as depicted in Figure 2. Note, however, that the results in Figure 2 are ***not necessarily*** correct.