Homework 2
Image Sampling and Filtering

- Deadline: Wednesday, 2019-11-27, 23:45
- Submission: Upload your code on Cody Coursework™ Mathworks platform https://grader.mathworks.com/courses/9867-mdsp-ws-2019-20 according to the instructions given there for all problems marked with (Cx.y).
- For all problems marked with (Ax.y) refer to the quiz questions at https://www.moodle.tum.de/course/view.php?id=50238. There you can also find the homework rules and more information.
- Result verification: Check your functions as often as you want with Cody Coursework™.
- Lab session for MATLAB questions – see website, room 0943
- Notation: `variable name`, *`file name`*, `MATLAB_function()`

1 Image Sampling

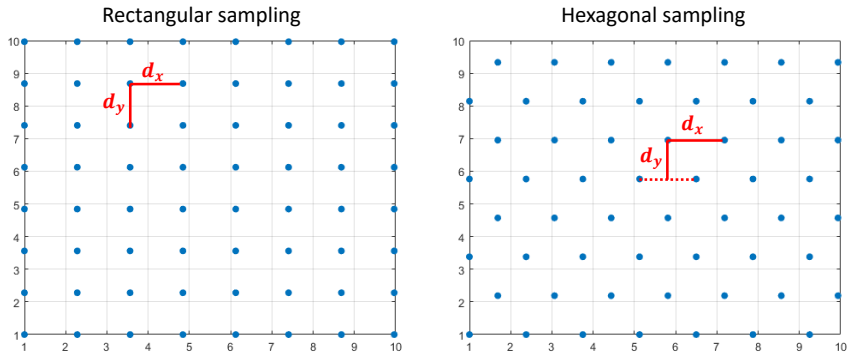

Figure 1: Sampling points for hexagonal and rectangular sampling grids

In this question we will work on image sampling with two different sampling grids, rectangular and hexagonal. In a hexagonal grid (Figure 1), the distances of a sampling point to all its 6 neighbors are equal, whereas in a rectangular grid there are 4 neighbors with distance a and 4 more neighbors with distance $\sqrt{2}a$. In this problem we investigate sampling effects on a natural image.

1. (C1.1) Generate a regular and rectangular sampling grid of dimensions $N \times N, N = 400$, which starts at $(1, 1)$ and spans (exactly!) the range $[1, 512]$ for x and y. Use `meshgrid()` and save the sampling coordinates as $N \times N$ matrices to ☞ `gridX` and ☞ `gridY`. Note the convention of `meshgrid()` to return sampling coordinates as matrices which resemble the spatial locations of the sampling points. *(5 points)*

2. (C1.2) Now you start working with the gray scale Lenna image, `im`. In order to find image intensity at non-integer coordinates, you need to use 2D interpolation: Create a smaller interpolated version of `im` of size $N \times N$ using `interp2()` with the grid from above and try the different interpolation options. As the resizing factor is small, skip the low-pass filtering step. Implement the corresponding function so that it performs interpolation using "linear" and "cubic" interpolation options and returns both images. Note: you do not need to implement the interpolation yourself, just reuse the built-in Matlab function. *(5 points)*

3. (A1.1) Describe how the resulting images (i.e. images after cubic and linear interpolation) differ and compare the required runtime, e.g. with `tic / toc`. Make sure to repeat interpolation for

10 times when performing the timing experiments and accumulate the timing. Otherwise, the implementation differences in different Matlab versions can lead to inconsistent results. *(5 points)*

4. (C1.3) Next, reconstruct the image in its original size 512×512 from the smaller `im_small`. You are given grid coordinates as computed from problem (C1.1). Images of square size are assumed. Use the `TriScatteredInterp()` class with "linear" interpolation this time. It works similar to `interp2()`, but can also handle non-regular grids. The resulting function will return the reconstructed image `im_rec`. *(6 points)*

5. (C1.4) Generate a horizontally aligned hexagonal grid with the first sampling point located at $(1, 1)$. The horizontal sampling distance is $d_x = \frac{511-1}{372-1}$ and the vertical sampling distance is $dy = \sqrt{3} \cdot 0.5 \cdot dx$, which are depicted in Figure 1. The sampling locations must span horizontally (exactly!) $x \in [1, 511 + \frac{1}{2}d]$ and vertically $y \in [1, 512[$. The function you implement should return the grid coordinates stored in two matrices ☞ `hexaX` and ☞ `hexaY`. *(10 points)*

   Hints: This hexagonal grid will have a total of 159960 sampling points. Create it as follows:

   - Generate the points of the grid first using `meshgrid()` and with the given d_x and d_y.
   - Shift each second raw of the calculated grid by $0.5d_x$ to obtain the hexagonal grid.
   - Verify that the grid fulfills the given constraints and that the distances of each point to all of its six neighbors are equal.

6. (C1.5) Similar to step (2), create a smaller version of `im`, interpolated at the hexagonal sampling positions using `interp2()`. Save the resulting image to ☞ `im_h`. After that repeat the reconstruction from step (4) with the hexagonally sampled image and return the resulting image. Make sure that you specify the correct coordinates of your input data for `TriScatteredInterp()`. *Hint:* Interpolation function returns NaN for border points, where no interpolation can be made. Simply ignore this fact for this problem. *(8 points)*

7. (C1.6) After you perform reconstruction, you are normally interested to know what reconstruction quality is achieved. For this, you usually need to compute PSNR of the difference image. In this problem, implement a function to compute PSNR of the difference image between the original and reconstructed versions. Remove a border of 5 px on all sides to neglect border effects. Save the values you obtain to ☞ `psnr_diff`. For images with a maximum intensity of 1, the PSNR is calculated from the mean squared error (MSE) as follows:

$$\text{PSNR [dB]} = 10 \log_{10} \frac{1}{\text{MSE}}, \quad \text{MSE} = \frac{1}{N} \sum_{x,y \in \mathcal{I}} (i_1(x, y) - i_2(x, y))^2,$$

   where N is number of considered pixels in the image. Note: Keep in mind that Matlab has a function with a name `psnr()`, therefore avoid calling your variables with the same name. This leads to errors that are hard to trace back. *(5 points)*

8. (A1.2) Analyze PSNR computed in step (7) for both reconstructed images. Which image has a better (higher) PSNR? How do you explain this difference? *(5 points)*

2 Zone Plate

A circular zone plate is an image of a sinusoidal circular wave which increases in frequency with the distance to a central point. It is useful to analyze effects of filtering, sampling and aliasing. In this problem, you will create a zone plate with the central point at the top-left pixel and the maximum possible frequency along the image diagonal in the bottom-right corner (see Figure 2). Using a linear frequency chirp $f(t) = kt$, the circular zone plate image is calculated as follows:

$$I_{zp}(x,y) = \frac{1}{2} + \frac{1}{2}\cos\left[2\pi \int_0^{d(x,y)} f(t)\,dt\right] = \frac{1}{2} + \frac{1}{2}\cos\left(2\pi\frac{k}{2}d(x,y)^2\right)$$

with distance from central point $d(x,y) \in [0, d_{max}]$ and constant k which adjusts the maximum frequency.

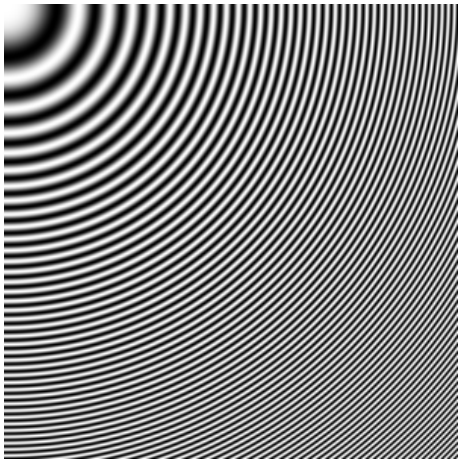

Figure 2: Image of a zone plate

1. (C2.1) Write a function to generate a $N \times N$ image matrix ☞ `D` whose entries represent the Euclidean distance in pixel units of the respective pixel to the top-left pixel. For the top-left element (`D(1,1)` in MATLAB) it is 0, for the bottom-right element (`D(N,N)` in MATLAB) $d_{max} = \sqrt{2}(N-1)$. Verify image `D` and display it for instance with `imagesc()`. Do not use for-loops – the functions `repmat()` or `meshgrid()` provide helpful initializers to calculate `D`. The maximum frequency of the zone plate is observed in the bottom right corner. Calculate ☞ `k` such that the phase change along the last two diagonal pixels is exactly π. Note that the distance of two pixels along the diagonal is $\sqrt{2}$. Now calculate the zone plate image ☞ `Izp` according to the given formula. *(10 points)*
2. (A2.1) Sub-sample `Izp` with factors 2:1 and 4:1 using sub-matrix addressing. Display the resulting images and explain the effects you observe. Make sure you visualize the image with 1:1 scaling. *(6 points)*

3 Image Filters

In this problem, you will work with several 2D filters. Their effects are observed on natural images and on the zone plate image.

1. (A3.1) Use the `fspecial()` function to generate image filters of types "gaussian","laplacian", "log", "prewitt". Work with sizes $3 \times 3, 8 \times 8$ and 16×16 as well as with different parameters. Plot the filter kernel with `surf()`. Then, analyze the frequency response of the generated filters using `freqz2()`. How does the filter size influence the frequency response? *(9 points)*

2. (C3.1) Find out how `freqz2()` generates its plot by inspecting its source code and write a function to do the same manually using the `fft2()` function. The function should return z values ☞ `prew_z`, same as what `freqz2()` plots as z-values for the "prewitt" filter (assume default options for `fspecial()` and `freqz2()`). The top-left element of `prewitt_z` corresponds to $(x, y) = (-1, -1)$, this way zero frequency component is located at the center of the spectrum. *(5 points)*

3. (C3.2) Write a function to filter the image with a Gaussian filter (16×16, $\sigma = 3$) and "replicate" border processing. *(5 points)*

4. (A3.2) Load image *`pears.jpg`* as double and filter it with the created filters (using `imfilter()`), without introducing any shift or scaling. Can you explain the observed effect from the frequency response of the filter? What is the effect of prewitt filter? What is the effect of a large filter size? How do you have to choose the size and σ value of a Gaussian low-pass filter to blur out fine image details, such as the marks on the pears? *(6 points)*

5. (C3.3) Now you will work with the zone plate image `Izp` from the problem C2.1 (1). Write a function that filters `Izp` with a Gaussian (16×16, $\sigma = 3$, boundary option "symmetric") and subsequently subsamples it with 2:1. The function should output the main diagonal of the resulting image. The second output variable is diagonal of non-filtered image with subsampling 2:1. *(5 points)*

6. (A3.3) Compare plots of the diagonals from the previous problem (5). How do they differ? What happens if you use MATLAB's default boundary option instead of the "symmetric" option? Compare the damping you observe with the frequency response of the Gaussian filter. *(5 points)*