Image Processing and Pattern Recognition

# Assignment 1

October 8, 2012

**Submission:** Deliver written solutions at the beginning of class on October 29, 2012. Send report and implementation (zipped) to `bvme@icg.tugraz.at`.
**Deadline:** Written solutions to task 1 (interpolation) - October 29, 2012, 14:15; Report and Matlab implementation (zipped) - October 29, 2012, 23:59

## 1 Interpolation (5 points)

Interpolation is necessary every time you rotate, translate or up-sample an image. In this task you should consider different interpolation algorithms.

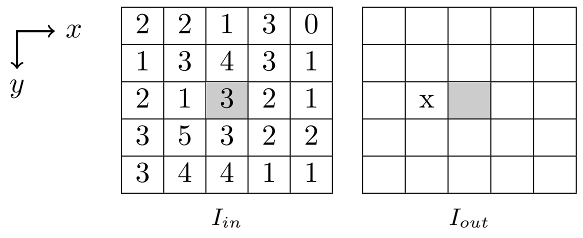
Assume that the given input image $I_{in}$ is rotated $-20°$. Calculate the value marked with an x in the output image $I_{out}$ (see Figure 1). Use

1. Nearest neighbor interpolation
2. Bilinear interpolation $g(x) \cdot g(y)$ where

$$g(x) = \begin{cases} 1 - |x| & \text{if } |x| \leq 1 \\ 0 & \text{else} \end{cases} \tag{1}$$

3. Bicubic interpolation $h(x) \cdot h(y)$ where

$$h(x) = \begin{cases} 1.5\,|x|^3 - 2.5\,|x|^2 + 1 & \text{if } |x| \leq 1 \\ -0.5\,|x|^3 + 2.5\,|x|^2 - 4\,|x| + 2 & \text{if } 1 < |x| \leq 2 \\ 0 & \text{else} \end{cases} \tag{2}$$



$I_{in}$:

| | | | | |
|---|---|---|---|---|
| 2 | 2 | 1 | 3 | 0 |
| 1 | 3 | 4 | 3 | 1 |
| 2 | 1 | 3 | 2 | 1 |
| 3 | 5 | 3 | 2 | 2 |
| 3 | 4 | 4 | 1 | 1 |

$I_{out}$:

| | | | | |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | x | ▨ | | |
| | | | | |
| | | | | |

Figure 1: Input and output image. The gray pixel defines the origin.

**Note:** All calculation steps should be clearly stated. You can either hand in your written solutions at the beginning of class (October 22) or include them in your report.

Now, implement the three algorithms in Matlab. Do not use the built-in functions like `imrotate`! Choose an arbitrary gray image and perform three rotations:

- $10^\circ$ clockwise
- $10^\circ$ clockwise
- $20^\circ$ counter-clockwise

Describe the three algorithms and the differences between them. Put the output images in your report and compare the images both visually and by means of their *peak signal-to-noise ratio* PSNR.

The PSNR for two images $I, J$ of size $m \times n$ is defined as:

$$\text{PSNR} = \begin{cases} \infty & \text{iff } I = J \\ 10 \cdot \log_{10}\left(\frac{\max^2}{\text{MSE}}\right) & \text{else} \end{cases} \tag{3}$$

where max is the maximum possible pixel value (i.e. 255 for a discrete image) and the MSE is the mean squared error defined as

$$\text{MSE} = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}\left(I(i,j) - J(i,j)\right)^2 \tag{4}$$

The higher the PSNR of two images, the more they resemble.

**Note:** Due to rotation parts of the images are clipped, if you leave the image size the same. You do not have to increase the image size such that the whole image can be displayed. Be careful if you compare your results to the original image! Take only those parts of the images for estimating the PSNR which do not contain new (black) pixels.

## 2 Denoising (5 points)

Image denoising is one of the classical problems in the field of image processing. In this task you have given three different pictures with three different types of noise (`noise_butterfly.jpg, noise_elephant.jpg, noise_gorilla.jpg`). One picture has been contaminated with additive Gaussian noise, one with salt and pepper noise and one with speckle noise.
Your task is to implement and apply three different kinds of filters with different kernel/support sizes and report the performance (PSNR) of each filter per image.

Two filters are applied via convolution (implement it on your own). The first filter kernel is a simple blur/average filter. The second filter kernel is a Gaussian kernel $h$ and can be computed as follows:

$$h(x, y) = \frac{1}{Z} e^{-\frac{x^2 - y^2}{2\sigma^2}} \tag{5}$$

$$Z = \sum_{x,y} e^{-\frac{x^2 - y^2}{2\sigma^2}} \tag{6}$$

where the proper size of the quadratic filter kernel is $n = 4\lceil\sigma\rceil + 1$.
As third filter you should implement a simple median filter.

Report the runtime of the Gaussian filter and compare it to the built-in function `imfilter`. Give two reasons how you could improve the runtime of your filter!

**Bonus task (5 points)** Load the file `bonus.png`. You have a distorted image as displayed in Figure 2. What is the cause for the stripe pattern? How can you get rid of the stripes? Give a short explanation and reconstruct the image. Compare the reconstructed image to the original image `butterfly.jpg` (PSNR).

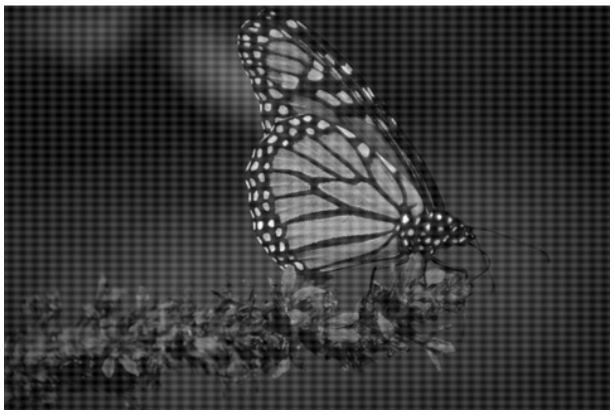


Figure 2: Distorted image

## 3 Canny Edge Detector (5 points)

The Canny edge detector is a commonly used edge detector in computer vision. The three optimality criteria for this detector are:

- Low error rate
- Well-localized edge points
- One response

In this task you have to implement the Canny edge detector. You can follow the steps below.

- Smooth the image with a Gaussian filter to remove noise.
- Calculate the gradient of the image (i.e. use a Sobel kernel for x- and y-derivatives) and derive the norm and angle of the gradient.
- **Non-maximum suppression:** Apply non-maximum suppression to the gradient images. Therefore you have to find the directions which are closest to the estimated angles. The four principal directions are $[-\infty, \frac{\pi}{4}], (\frac{\pi}{4}, \frac{\pi}{2}], (\frac{\pi}{2}, \frac{3\pi}{4}], (\frac{3\pi}{4}, \pi]$ (note: $\phi = \phi - \pi$, hint: `histc`).
  Then compare the edge strength of each pixel to the neighbor pixels in the positive and negative gradient direction (i.e. if the gradient direction of the pixel is $\frac{\pi}{2}$ then compare the edge strength with the pixels to the north and south). If the edge strength is not the local maximum, suppress the edge value (i.e. set the value to 0).
- **Double thresholding:** The Canny edge detector uses two thresholds. Edge values smaller than the lower threshold are set to 0, edge values higher than the upper threshold are set to 1, and edge values between those two thresholds are set to 0.6.
- **Hysteresis:** After the above step you have strong edges (i.e. edge value is 1) and weak edges (i.e. edge value is 0.6). Now we want to compute the final binary edge image. Strong edges are immediately included in the result. A weak edge should be included in the final image, if a strong edge is in its 8-neighborhood. Note that this is an iterative process.

Your report should include the output images and a description of each step. More information on the Canny edge detector can be found in [1].
**Note:** Handle the image borders by replicating the border pixels.

## References

[1] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing.* Pearson Prentice Hall, 3rd edition, 2008. ISBN 978-0-13-505267-9.