Computer Vision 1 - Filters

February 18, 2017

All the files should be zipped and sent to **computervision1.uva(at)gmail.com** before **27-02-2017** , **23.59** (Amsterdam Time). Remember to include your report. The first part is optional, you do not need to submit it.

1 Gaussian Filters (Optional)

In this section, you will be working with Gaussian filters and applying them to images, exploiting their separability property.

1.1 1D Gaussian Filter

The 1D discrete Gaussian filter is defined as follows:

$$G_\sigma = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{x^2}{2\sigma^2}), \tag{1}$$

where σ is the variance of the Gaussian.

Implement a MATLAB function **gauss** that computes G given σ and kernel_size as parameters. Your function should look like:

```
function G = gauss(sigma,kernel_size)
...
end
```

1.2 2D Gaussian Filter

One of the most important properties of 2D Gaussian kernels is separability. Therefore, convolving an image with a 2D Gaussian is equivalent to convolving the image twice with a 1D Gaussian filter, once along the x-axis and once along the y-axis **separately**.

Implement a MATLAB function **gaussConv** that performs the 2D Gaussian convolution using the **gauss** function implemented in 1.1. Your function should look like:

```
function imOut = gaussConv(image,sigma_x,sigma_y,kernel_size)
...
end
```

Hint: Use MATLAB function conv2.

(a) Visualize the results of your function using (image1.jpeg) with different values for kernel_size, sigma_x and sigma_y.

(b) Compare your implementation of 2D Gaussian convolution (using two 1D operations) to MATLAB's implementation using built-in function (**fspecial**). Use (image1.jpeg) to visualize the difference between your resulting image and MATLAB's one.

1.3 Gaussian Derivative

The Gaussian first order derivative is given by:

$$\begin{aligned} \frac{d}{dx}G_\sigma &= \frac{d}{dx}\left(\frac{1}{\sigma\sqrt{2\pi}}\exp(\frac{x^2}{2\sigma^2})\right) \\ &= -\frac{1}{\sigma^3\sqrt{2\pi}}x\exp(-\frac{x^2}{2\sigma^2}) \\ &= -\frac{x}{\sigma^2}G_\sigma \end{aligned} \tag{2}$$

Implement a MATLAB function **gaussDer** that computes the first order derivative and applies it on a given image. Your function should look like:

```
function [imOut Gd] = gaussDer(image,G,sigma)
...
end
```

2 More on Filters

2.1 Gaussian Versus Box

Implement a MATLAB function **denoise** that removes the noise from a given input image. The function will denoise the image by either applying median filtering or box filtering methods. Your function should look like:

```
function imOut = denoise(image, kernel_type, kernel_size)
...
end
```

(a) Using your implemented function **denoise**, try denoising (image2.jpeg) by applying the following filters:

  (i) Box filtering of size: 3x3, 5x5, 7x7 and 9x9.

  (ii) Median filtering with size: 3x3, 5x5 and 7x7.

(b) Discuss the effect of increasing the filter size.

(c) Which is better, box or median filters? Why?

2.2 Histogram Matching

Implement a function **myHistMatching(input,reference)**. The function takes two images as input and transforms the first image so that it has a histogram that is the same as the histogram of the second image. Your function should look like:

```
function imOut = myHistMatching(input, reference)
...
end
```

The function is expected to display the following 3 figures:

- Figure 1: Input image and its corresponding histogram in the same figure.
- Figure 2: Reference image and its corresponding histogram in the same figure.
- Figure 3: Transformed version of input image and its corresponding histogram in the same figure.

Apply this function so that the (input.png) is transformed to have the same histogram as the (reference.png).

2.3 Gradient Magnitude and Direction

Implement a MATLAB function **compute_gradient** that computes the magnitude and the direction of the gradient vector corresponding to a given image. The function should use the **Sobel** kernel, which approximates the 2D derivative of an image. Your function should look like:

```
function [im_magnitude,im_direction] = compute_gradient(image)
...
end
```

* **You're not allowed to use the MATLAB's *imgradient* built-in function.**

Test your function using (image3.jpeg), where you have to display the following 4 figures:

- Figure 1: The gradient of the image in the x-direction.
- Figure 2: The gradient of the image in the y-direction.
- Figure 3: The gradient magnitude of each pixel.
- Figure 4: The gradient direction of each pixel.

Hints:
* The gradient magnitude is defined as the square root of the sum of the squares of the horizontal (G_x) and the vertical (G_y) components of the gradient of an image, such that:

$$G = \sqrt{{G_x}^2 + {G_y}^2} \tag{3}$$

* The gradient direction is calculated as follows:

$$\theta = \tan^{-1}(G_x/G_y) \tag{4}$$

2.4 Unsharp masking

Implement a MATLAB function **unsharp** that highlights the image fine details. First, you have to smooth the original image using Gaussian filter with given kernel size and Sigma. Then, you need to subtract the smoothed image from the original image to get a high-passed version of the original image. Finally, you can get the final version of the image by strengthening the high-passed image k times (k is the weighting factor) and adding it to the original image.

Your function should look like:

```
function imOut = unsharp(image, sigma, kernel_size, k)
...
end
```

(a) Test your function using (image4.jpeg) and visualize your results.

(b) Explain the difference between using $k = 1$ and $k > 1$.

(c) What is the effect of changing the kernel size of the Gaussian filter on the resulting image?

(d) What are the common applications of unsharp masking?

2.5 Laplacian of Gaussian

Laplacian of Gaussian (LoG) can be computed by the following three methods:

- Method 1: Smoothing the image with a Gaussian operator, then taking the Laplacian of the smoothed image.
- Method 2: Convolving the image directly with LoG operator.
- Method 3: Taking the difference between two Gaussians (DoG) computed at different scales σ_1 and σ_2.

Implement a MATLAB function **compute_LoG** that performs the Laplacian of Gaussian. The function should be able to apply any of the above mentioned methods depending on the value passed to the parameter *LOG_type*. Your function should look like:

```
function imOut = compute_LoG(image, LOG_type)
...
end
```

(a) Test your function using (image1.jpeg) and visualize your results using the three methods.

(b) Discuss the differences between applying the three methods.

(c) In the first method, why is it important to convolve an image with a Gaussian before convolving with a Laplacian?

(d) In the third method, what is the best ratio between σ_1 and σ_2 to achieve the best approximation of the LoG?

(e) What are the common applications of the LoG filter?