# Computer Vision 1 - Assignment 2

**Ruben-Laurentiu Grab**
ruben.grab@student.uva.nl 11609923

**Andrei Marius Sili**
andrei.sili@student.uva.nl 11659416

## 1 Introduction

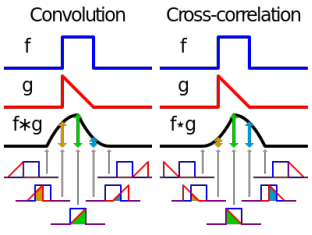
In this report we investigate image processing techniques such as the Gaussian and Gabor filters, edge detection using Laplacian of Gaussian, PSNR and foreground-background segmentation. We take an empirical approach, experimenting with different setups and parameters, reporting our results thoroughly. Whenever possible, we provide theoretical grounding for our findings.

## 2 Neighbourhood processing

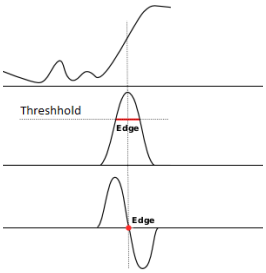
**Question 1.1** The basic difference between convolution and correlation is that the convolution process rotates the matrix by 180 degrees. The implications of this modification is that by doing so, you compute two totally different things. Theoretically, convolution are linear operations on the signal or signal modifiers, whereas correlation is a measure of similarity between two signals. In other words, when one wants to compute the similarity between two images, such as matching a template to an image, one uses cross-correlation. Convolution is used for image processing operations such as smoothing or edge detection. Moreover, another difference between the two is that convolution is associative. This is helpful, as in, rather than applying several filters one after the other: $(((a * b1) * b2) * b3)$, one can apply just one filter: $a * (b1 * b2 * b3)$.

Figure 1a helps us visualize the difference between these two operators. Although the images is drawn from a signal processing point of view, the principles are the same. We can observe that the $g$ mask is reversed(just how we rotate the *kernel*) for convolving, and the resulting signal(but can be extended to images, in our case) borrows features form both $f$ and $g$ signals.

**Question 1.2** Given the above mentioned difference between *correlation* and *convolution*, when we assume the mask matrix $h$ to be symmetrical about the two axes, thus the kernel and its rotation being identical, the correlation and convolution operators are equivalent.




(a) Convolution vs. Correlation




(b) Edge detection with first and second order derivative Gaussians

## 3 Low-level filters

### 3.1 Gaussian filters

**Question 2** A 2D Gaussian kernel is a separable kernel and can be be expressed as the product of two identical 1D Gaussian kernels. Given the fact that the convolution is associative, applying a 1D Gaussian kernel on the x and y axes separately would yield the same result as convolving once with a 2D Gaussian kernel.

But generally, a 2D convolution is more expensive than two 1D convolutions. When we are applying a 2D filter we are making for each pixel $N * N$ multiplications, whereas when applying twice a 1D filter on the $x$ and $y$ axes, for each pixel we are making for each pixel $2 * N$ multiplications. In other words, for a $M \times M$ image and a $N \times N$ kernel, we can reduce the complexity from $N^2M^2$ to $2NM^2$

**Question 3** Convolving the first derivative of the Gaussian kernel with an image, we get the edges on the local maximum values of the convolution. When applying the second derivative of a Gaussian, the nice result is that the edges are represented by the points where the convolution of the input image with the second derivative is zero(see figure 1b).

### 3.2 Gabor filters

**Question 4** The Gabor filter is a convolution kernel that responds to edges at changes of texture. It is computed as a Gaussian modulated with a complex sinusoidal carrier signal, controlled by several parameters. In experimenting with the parameters, we mainly look at the 'Kodi' image convoluted with the imaginary part since most of the features we are interested in for the purpose of this demonstration are oriented vertically (e.g. the edges between tiles).

**Lambda** $\lambda$. This parameter controls the wavelength of the sinusoidal signal carrier, which can be interpreted as the harmonic frequency of the signal. A higher $\lambda$ will widen the ellipses that generate the filter response. As such, increasing the wavelength will result in the filter responding more strongly to broader features (e.g. thicker edges). Figure 2 compares different values of lambda for $\theta = 0$, $\sigma = 1$, $\gamma = 0.5$, and $\psi = 0$. One can observe that the vertical edges generate the strongest response when the wavelength is set to $\lambda = 5.65$ and fade out for other values [1].

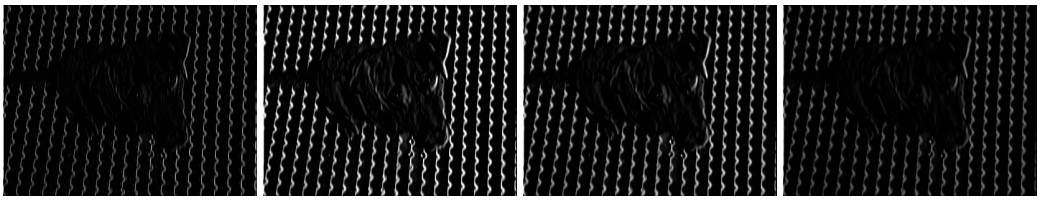
**Theta** $\theta$. This is likely the most important parameter of the Gabor filter. Mathematically, $\theta$ represents the orientation of the normal to the parallel stripes of the filter. This means that $\theta$ controls the orientation of the features to which the filter will respond most strongly. let us assume everything else equal, and orientation ranging between0 and $\pi/2$ [1]. An increment in $theta$ will result in the filter responding to identical features that are rotated to the right compared to the previous value of $\theta$. Figure 3 shows this effect. An orientation of $0$ responds to vertical edges, while an orientation of $\pi/2$ responds to horizontal edges and an orientation of $\pi/4$ responds to diagonal ones [2].

**Sigma** $\sigma$. This is the parameter that controls the width of the Gaussian envelope. A higher sigma will result in a more diffused response, while a lower sigma will result in a more concentrated response. It controls the radius of the convolution kernel. We can envision this by observing that a Gaussian filter blurs the image to which the signal carrier is applied to get a response. The higher $\sigma$ is, the higher the blur, resulting in a diffused response. Figure 4 shows filter of varying envelope sizes with $lambda = 5.65$, $\theta = \pi/4$, $\gamma = 0.5$, and $\psi = 0$.

**Gamma** $\gamma$. This parameter controls how elliptic the Gaussian function is. A value close to $0$ will result in an elongated filter, while a value close to $1$ will result in a circular envelope.

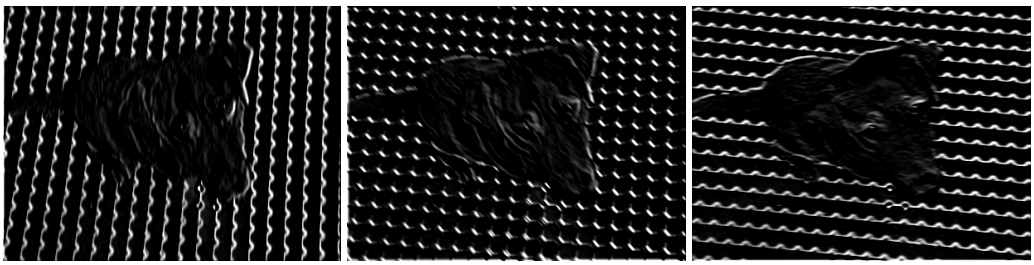
**Psi** $\psi$. This parameter can be seen as an offset of the signal. It controls the shift of the sinusoidal from the center. Assuming a measurement in radians, $0$ means that the max and min values of the signal will be in the center of the kernel, while $\pi/2$ gives max and min values in a position shifted from the center.

**Question 5** **Theta** $\theta$. In this experiment we vary $theta$ between $-\pi/2$ and $\pi/2$, holding constant the other parameters at $\lambda = 2.82$ $\sigma = 1$ $\gamma = 0.5$ $\psi = 0$. Figure 5 shows the different responses. The resulting filters are then applied to image 'Robin-2'. As noted previously, depending on the orientation chosen the filter responds more strongly to edges that are oriented at a similar angle to



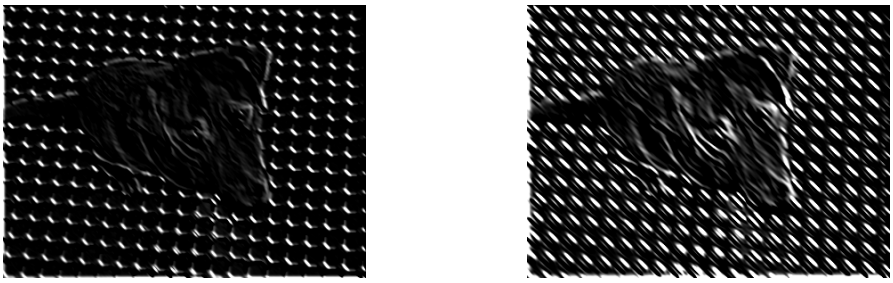
(a) $\lambda = 2.82$ (b) $\lambda = 5.65$ (c) $\lambda = 11.31$ (d) $\lambda = 22.62$

Figure 2: Response for varying $\lambda$ ($\theta = 0$ $\sigma = 1$ $\gamma = 0.5$ $\psi = 0$)



(a) $\theta = 0$ (b) $\theta = \pi/4$ (c) $\theta = \pi/2$

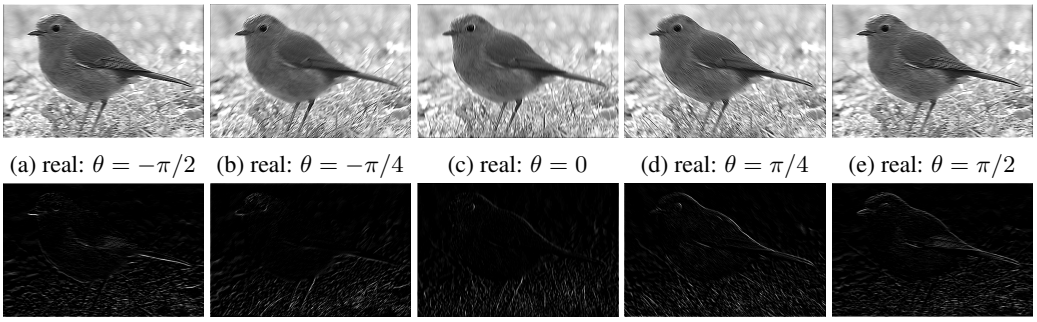
Figure 3: Response for varying $\theta$ ($lambda = 5.65$ $\sigma = 1$ $\gamma = 0.5$ $\psi = 0$)



(a) $\sigma = 1$ (b) $\sigma = 2$

Figure 4: Response for varying $\sigma$ ($lambda = 5.65$ $\theta = \pi/4$ $\gamma = 0.5$ $\psi = 0$)

$\theta$. We add to the previous section by noting that ranging $\theta$ over negative values as well as positive, results in a strong response to diagonals in both directions, as well as vertical and horizontal ones. This can be seen when looking at images (d) and (h). One observation that we cannot explain is why the real part of the filter gives such uninformative responses. This is the case for all wavelengths tested, but not for a Gaussian spread higher than 1. We explore this topic in further detail.

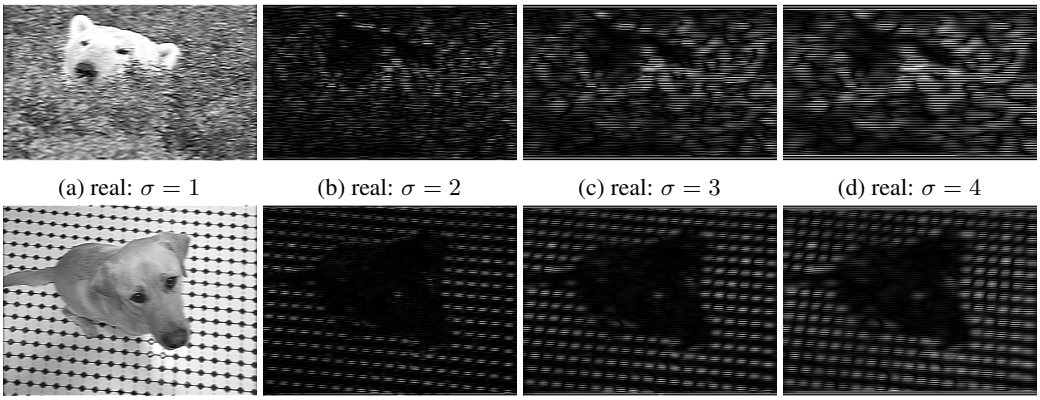


(a) real: $\theta = -\pi/2$ (b) real: $\theta = -\pi/4$ (c) real: $\theta = 0$ (d) real: $\theta = \pi/4$ (e) real: $\theta = \pi/2$

(f) imag: $\theta = -\pi/2$ (g) imag: $\theta = -\pi/4$ (h) imag: $\theta = 0$ (i) imag: $\theta = \pi/4$ (j) imag: $\theta = \pi/2$

Figure 5: Response for varying $\theta$ ($lambda = 2.82$ $\sigma = 1$ $\gamma = 0.5$ $\psi = 0$)

**Sigma** $\sigma$. We vary $\sigma$ between $0.5$ and $4$, keeping others constant at $lambda = 2.82$, $\theta = -\pi/4$, $\gamma = 0.5$, and $\psi = 0$. We apply the filter bank on the images 'Polar' and 'Kodi' and observe two interesting phenomena. First of all, the real part of the filter is now starting to detect features. Second, and most importantly, as $\sigma$ increases, the increasingly responds to blob features instead of edges, because of the increase in the size of the Gaussian envelope.With higher $\sigma$ , features such as the dog's eyes trigger responses in the filter. However, in the case of the 'Polar' image, these blobs do not represent any meaningful information, since there are no discernible blob shaped feature in this image. Figure 6 shows the real responses of the filter for both images side by side for comparison.

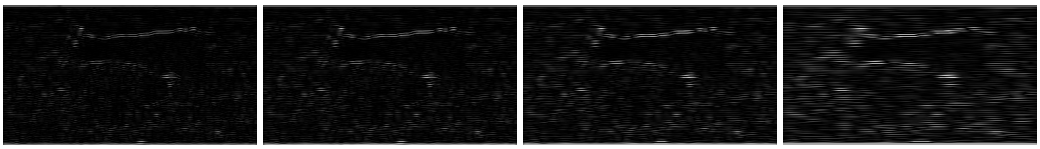


(a) real: $\sigma = 1$ (b) real: $\sigma = 2$ (c) real: $\sigma = 3$ (d) real: $\sigma = 4$

(e) real: $\sigma = 1$ (f) real: $\sigma = 2$ (g) real: $\sigma = 3$ (h) real: $\sigma = 1$

Figure 6: Response for varying $\sigma$ ($lambda = 2.82$ $\theta = -\pi/2$ $\gamma = 0.5$ $\psi = 0$)

**Gamma** $\gamma$. As mentioned earlier, this parameter controls the elipiticity of the the Gaussian envelope. The result of varying $\gamma$ on the imaginary filter applied to the 'Cows' image is displayed in figure 7. It can be seen that for a gamma of 1, the envelope is circular, and as the proportions get more far apart (i.e gamma decreases), the envelope gets stretched in in the direction of $\theta$.



(a) real: $\gamma = 1$ (b) real: $\gamma = 0.75$ (c) real: $\gamma = 0.50$ (d) real: $\gamma = 0.25$

Figure 7: Response for varying $\gamma$ ($lambda = 2.82$ $\theta = \pi/2$ $\sigma = 2$ $\psi = 0$)

# 4 Applications in image processing

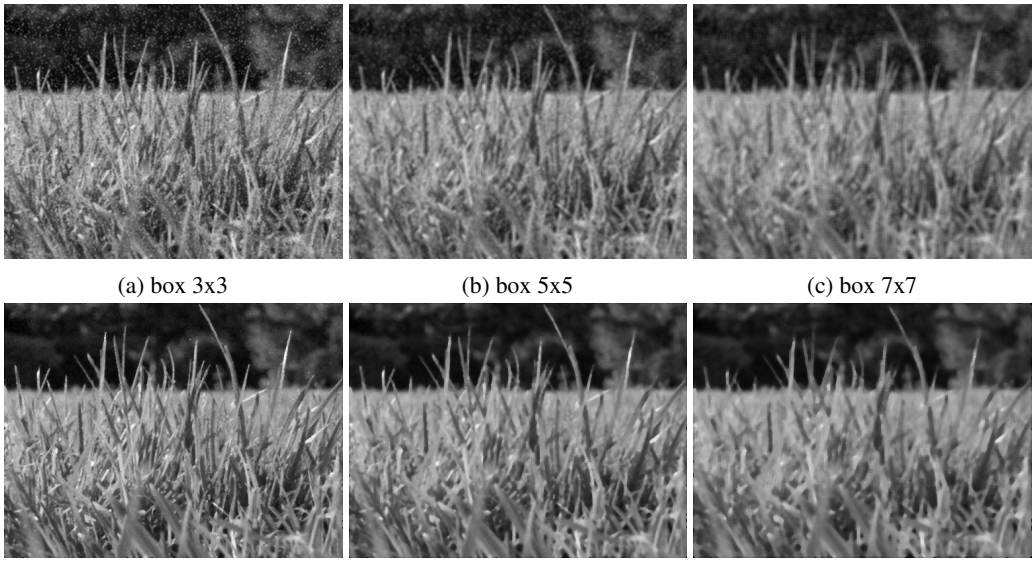
## 4.1 Noise in digital images

## 4.2 Image denoising

**Question 6.1** Indeed, after implementing our version of the algorithm that computes the peak signal-to-noise ratio (PSNR) and computing the *psnr* between images "*image1_saltpepper.jpg*" and "*image1.jpg*", the algorithm yielded the value of $16.1079$ dB.

**Question 6.2** Using our implementation of the algorithm to compute the *psnr* between "*image1_gaussian.jpg*" and "*image1.jpg*", we got the value of $20.5835$ dB.

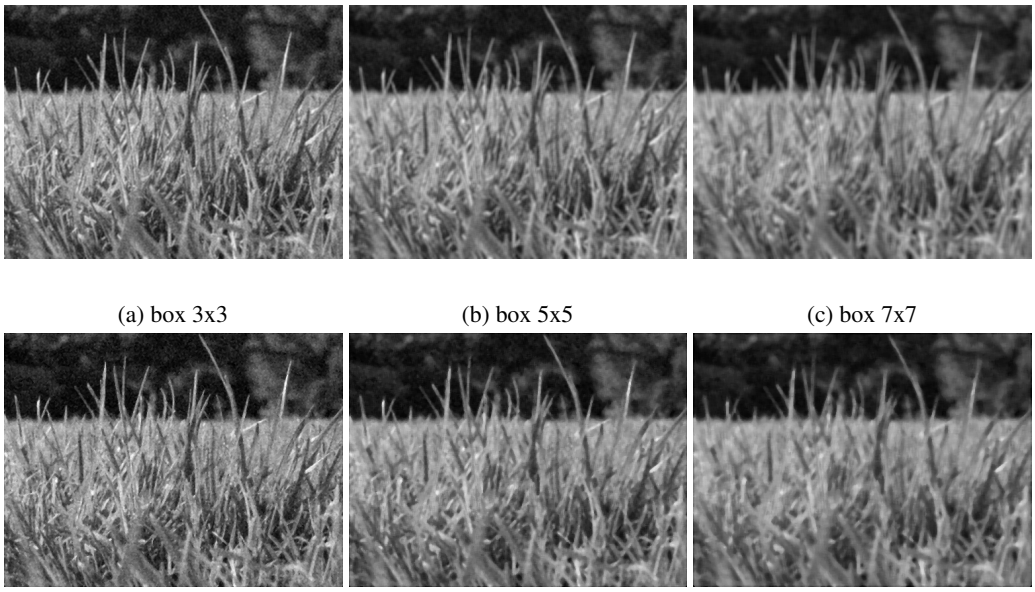
**Question 7.1, 7.2** Peak signal-to-noise ratio(*PSNR*) is a measure that is used to compute the quality of reconstruction of lossy compression. In our case, we use it to compute how similar the denoised image is to the original image. Although a higher PSNR generally indicates that the reconstruction is of higher quality, it is not always the case. Moreover, is only conclusively valid when it is used to



(a) box 3x3 (b) box 5x5 (c) box 7x7

(d) median 3x3 (e) median 5x5 (f) median 7x7

Figure 8: Denoising of *image1_saltpepper.jpg*, with box and median filters of different sizes



(a) box 3x3 (b) box 5x5 (c) box 7x7

(d) median 3x3 (e) median 5x5 (f) median 7x7

Figure 9: Denoising of *image1_gaussian.jpg*, with box and median filters of different sizes

compare results from the same content, or similar setups. Having the same base, noise free image and using the same denoising algorithms but with different window sizes per type of noise added, we meet these conditions.

We ran an experiment in which we tested two denoising algorithms using two different approaches, namely box and median filtering, on two types of noises: salt-and-pepper noise and Gaussian noise. In addition to that, we also varied the size of the window form 3 by 3 to 7 by 7. When comparing the effect the window size has on the quality of denoising, we observe a decreasing trend in the value of *psnr* as we increase the window size. This happens because for each pixel in the image, the algorithm takes information from more surrounding pixels as we increase the window size . For some images,

Table 1: PSNR values for different images, kernels and kernels sizes. It has to be mentioned that the denoised images were saved as **.png* files. When saving the images under **.jpg* files, the *psnr* scores are overall higher, but the trends remain the same. This shows that there is difference in the compression using *png* and *jpg*, and *psnr* algorithm can measure it.

| | | 3x3 | 5x5 | 7x7 |
|---|---|---|---|---|
| image1_saltpepper.jpg | box | 23,3941 | 22,6410 | 21,4220 |
| | median | 27,6875 | 24,4957 | 22,3722 |
| image1_gaussian.jpg | box | 26,2326 | 23,6610 | 21,9441 |
| | median | 25,4567 | 23,7983 | 22,0765 |

increasing the window size will yield better results, but because our image is fine grained and has small areas, taking more information from the surroundings is not a good strategy. Table 1 shows that invariant of the noise added to the original image and type of algorithm used, increasing the windows size negatively influences the *psnr* value.

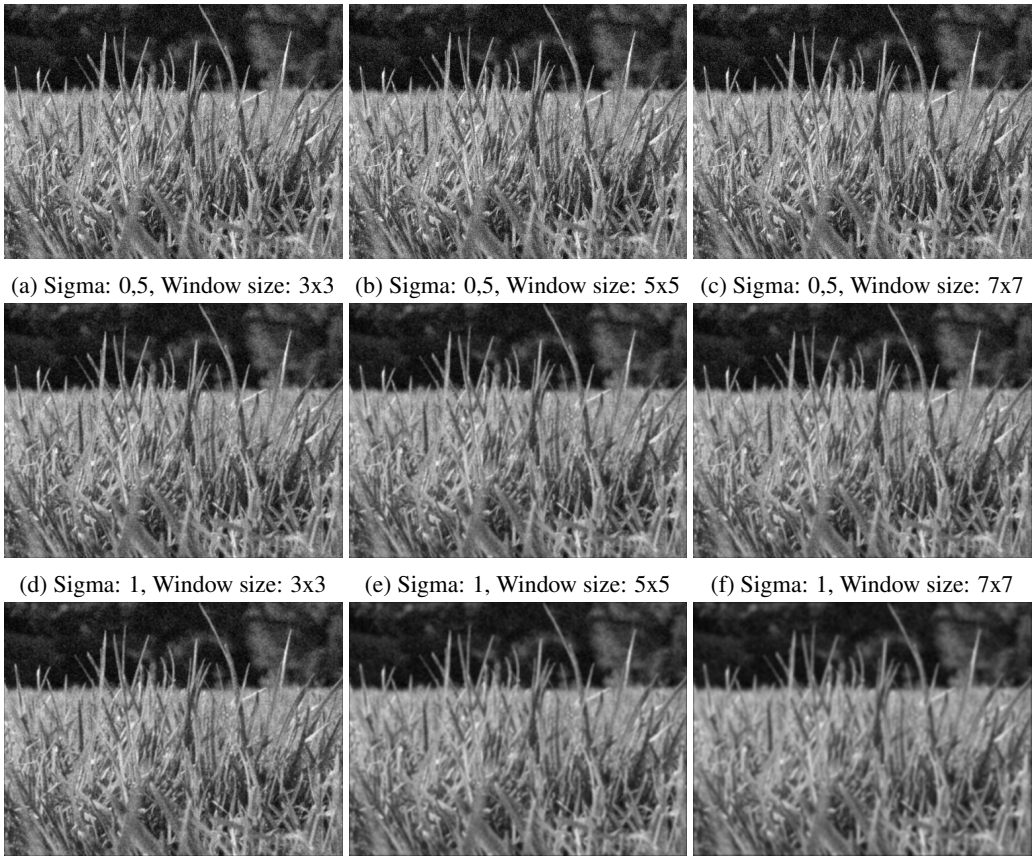
**Question 7.3** When comparing the results for the two types of noise used and the types of filters, we observe that there is not a clear *gold* filter that can outperform the other filters regardless of the noise used. Table 1 shows us that using a median filter for salt-and-pepper noise is a better approach. The reason for this lays in the way the media filter woks, and the properties of the salt-and-pepper noise. The median filter chooses for each pixel the median value of all the pixels in the window size. The pixels that are from salt-and-pepper noise will tend to be towards the ends of the sorted pixels in the window size, thus having a lower probability of being chosen as the output value. In addition, usually the pixels that are not from the noise tend to have roughly the same value. Given the mentioned particularities of the filter and the noise pixels, is is also clear to see why using a smaller window size yields better results. When using a box filter, the noise is taken into consideration when taking the average of pixel values, and have greater influence on the output pixel. The results of the experiment regarding the salt-and-pepper noise can be seen in figure 8.

When looking a the Gaussian noise, we see a different pattern. The best approach is to use a box filter and the window size small. However, the difference between using a box and a median filter is relatively small when trying to correct Gaussian noise, compared to when correcting salt-and-pepper noise. From table 1 we observe that for Gaussian noise, the average of values and the median value tend to be relatively closer than when using salt-and-pepper noise. The results of the experiment regarding the Gaussian noise can be seen in figure 9.

**Question 7.4** We also run am experiment in which we varied the sigma parameter and also the window size. For sigma we used sequentially $0, 5, 1$ and $2$, and the windows size was set to $3$ by $3$, $5$ by $5$ and $7$ by $7$. the results in terms of the psnr value of the performance of the algorithm with each of the 9 different setups can be seen in table 2. We observe that generally, a smaller window size yields better results. The results in the table are also validated by the visuals in figure 10.

**Question 7.5** Following the experiment we performed, we also observed that invariant of the window size, sigma has a great impact on the quality of the image and on the *psnr* score between the original the denoised images. Looking at the images in figure 10 we have a visual cue of how sigma affects the image. We notice that a value too high for sigma blurs image too much, and a value too small sharpens it. This is to no surprise, if we think about what sigma does: a small value of sigma puts more emphasis on the pixels in the center of the window, and a large sigma takes more into consideration the pixels further from the center as well.

**Question 7.6** When using median filtering, we replace the pixel we are currently on with the median value of all pixels in the window. Gaussian and box filters work differently: they both take the average of the values in the window span, with the mention that for Gaussian, this average is weighted. Depending on the value of sigma, the Gaussian filter puts more or less weight on the pixels around the center, relatively to the pixels situated towards the edge of the window. If two filtering methods give a PSNR in the same ballpark there can indeed be seen a difference. If we look for example in table 1 at the *psnr* for the denoised *image1_gaussian.jpg* for a windows size of 5 by 5 for both box and median filters, although the psnr values are almost equal, we can clearly see a



(a) Sigma: 0,5, Window size: 3x3 (b) Sigma: 0,5, Window size: 5x5 (c) Sigma: 0,5, Window size: 7x7

(d) Sigma: 1, Window size: 3x3 (e) Sigma: 1, Window size: 5x5 (f) Sigma: 1, Window size: 7x7

(g) Sigma: 2, Window size: 3x3 (h) Sigma: 2, Window size: 5x5 (i) Sigma: 2, Window size: 5x5

Figure 10: Gaussian filter applied on a image, with 3 different setups for Sigma and Window size.

Table 2: *PSNR* values for a Gaussian filter. The row headers represent the values for *Sigma*, and the column headers represent the *Window size*. Denoised image were stored as **.png*

| | 3x3 | 5x5 | 7x7 |
|---|---|---|---|
| 0,5 | 24.2888 | 24.2954 | 24.2954 |
| 1 | 26.6024 | 26.1597 | 26.0981 |
| 2 | 26.1467 | 24.2152 | 23.2128 |

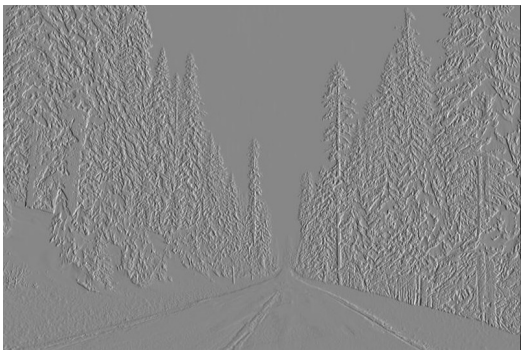
qualitative difference when we look at figure 9. Of course, it is in the eye of the viewer to say which one is better than the other.

### 4.3 Edge detection

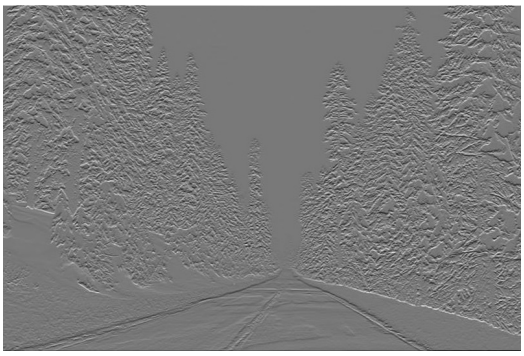
### 4.4 Question 9

The results of the three methods are shown in figure 12. We see that all give similar results, with only minor differences. For example, a minor difference can be seen between method 1 and 2 and between 2 and 3 with respect to intensity of the flat areas of the image. What is important though is that the response to the edges seems to be constant across the 3 images.

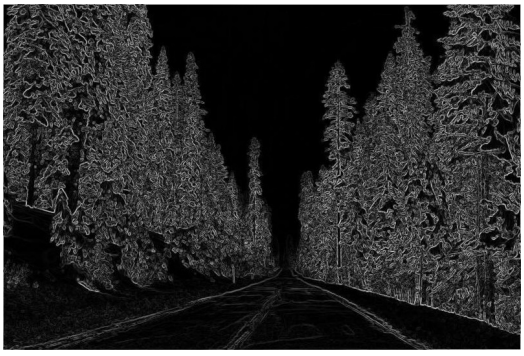
**Question 9.1, 9.2** The similarity in edge detection was to be expected between method 1 and 2. This is because the two methods are in fact equivalent. Method 1 first smooths the image using the Gaussian kernel and then applies the Laplacian kernel. Method 2 applies the LoG filter directly, which is the result of convolving the Gaussian kernel with the Laplacian kernel. Because convolution is an associative operation, the responses of the 2 methods should be identical. It is worth mentioning



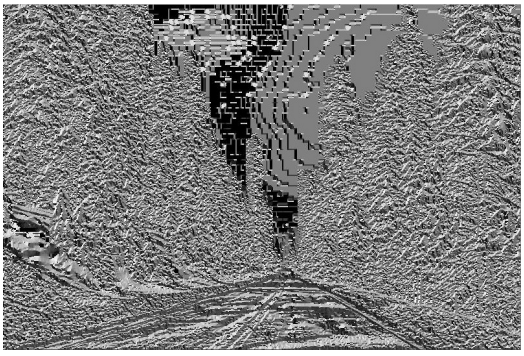
(a) The gradient of the image in the x-direction.



(b) The gradient of the image in the y-direction.



(c) The gradient magnitude of each pixel.



(d) The gradient direction of each pixel.

Figure 11: Question8: The gradients of *image2.jpg*

though, that the second method is much more efficient, because the two kernel are much smaller in size, and the LoG kernel can also be precomputed for all images. Comparing method 2 and 3, we note that that the DoG method is an approximation of the LoG method, given a proper choice of the standard deviation of the two Gaussians. Both reduce the image's high-frequency components, but the difference in the reduction power generated by different value of $\sigma$ make the filter respond most strongly to sharp edges, since those will have the highest difference in response between the two filters.

**Question 9.3** . Convolving with a Gaussian kernel has the effect of reducing any high frequency details of image. Since noise also has high spatial frequency, the Gaussian filter also reduces noise. This is important because the Laplacian filter enhances details of high spatial frequency and dampens others. The expectation is that, by first dampening noise, only the true features will be extracted by the Laplacian, since those are more resilient.

**9.4** Some tweaking was necessary to make the DoG a good approximating of the LoG. A quick brush of the literature suggests that the DoG method approximates the LoG method most accurately when the difference in magnitude between $\sigma_1$ and $\sigma_2$ is $\sqrt{2}$. That is $\sigma_1 = \sqrt{2}\sigma_2$ [3]. According to our experiments as well, this achieves the best approximation.

**9.5** Looking at the gradient of the first-order method(figure 11(c)) and at the gradients of the second-order derivative(figure 12) we can observe a slight change in the overall aspect of the edges. Using the algorithm based on the first-order derivative gives us edges that are not so well accentuated. Looking at the side of the road, but also at the trees on the side of the image, we observer that their edges are 'rounded'. On the other hand, looking at all three methods based on the second-order derivative we observe that the edges are more crisp and the road is more contoured. In other words, image 11(c) looks like a blurred version of the images in figure 12.

**9.6** Noticing that the previous methods respond to edges in all directions, we could use a Gabor filter with diagonal orientation. This should in principle eliminate most of the edges from the trees, apart from those that are oriented similarly to the edges of the road.

(a) Method 1 (b) Method 2 (c) Method 3

Figure 12: Responses of the LoG edge detector using 3 methods.

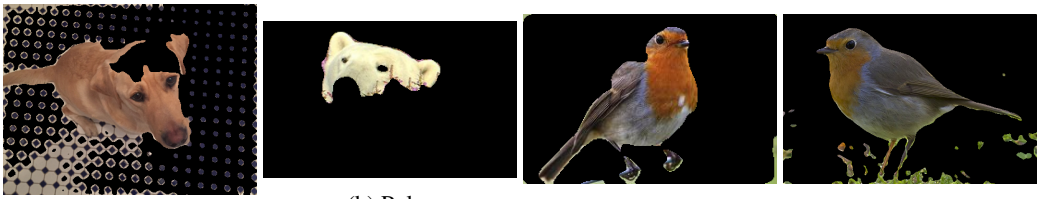
### 4.5 Foreground-background separation

**Question 10.1** We run the segmentation algorithm on all images with the default parameter ranges and smoothing with $\sigma = 3$. In general we observe decent results, with some shortcomings depending on the image. For example, much of Kobi is accurately identified as being in the foreground, but part of its scalp is treated as background. This could be due to the Gabor filter bank not capturing this particular orientation well, or the scalp being too similar to the tiles. It should also be noted that the black marks on the tiles and the area shadowed by Kobi are detected as foreground. Figure 13 shows only the foreground components.

The Polar image also shows promising results. Apart from the nose, the bear is perfectly segmented out of the background. This was to be expected to some extent, as the white fur contrasts strongly with the otherwise bland field of flowers.

Robin-1 is also interesting in that the bird is reasonably well picked out of the background, but some of the more bland coloured feathers are missing since they might blend too much with the background. The pole on which it is sitting is also missing from the foreground, which might be due to the orientation of the Gabor filters again.

One last thing to note is that the output of the segmentation sometimes show the foreground on the left and sometime on the right. This is expected due to the unsupervised nature of k-means. However, this poses a problem in online segmentation since one cannot be sure of the order of outputs.



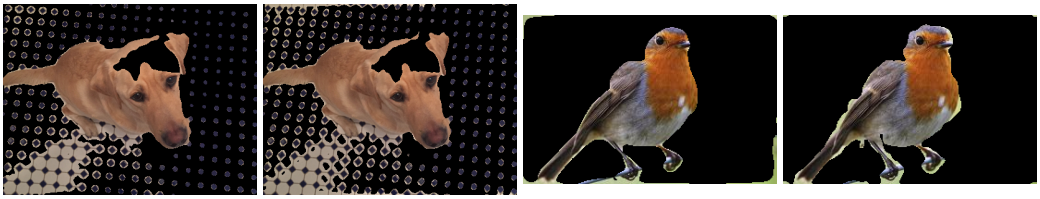
(a) Kobi (b) Polar (c) Robin-1 (d) Robin-2

Figure 13: Foreground segmentation of test images.

**Question 10.2** Tuning the parameters $\lambda$, $\sigma$, and $\theta$ rendered rather disappointing results. It seems that in our current setup, the choice of these parameters does not have substantially influence results. After some searching we chose to display two extremes. At one end, we segment the images with Gabor filters having only the minimum value of lambda, at an orientation angle of $\pi/8$. At the other end, we use the default range for lambda, a range of orientation from $0$ to $\pi$ with a step size of $\pi/8$ and Gaussian spread ranging from $0.5$ to $5$ with step size $0.5$. The results ca be seen in figure 14.

For the second modification, we expected to see more of the foreground captured, especially because we use more orientations, which should detect, for example, the pole on which the robin is sitting in image Robin-1. This is not the case however, with the result showing little difference to the other

cases. We are slightly baffled by this discovery and we are not sure what to make of it. The most we see is a slight better capture of the foreground when we use full ranges, but that comes at the expense of identifying more of the background as foreground as well.



(a) Kobi Full Range (b) Kobi No Range (c) Robin-1 Full Range (d) Robin-1 No Range

Figure 14: Varying parameters

**Question 10.3** Our experiments show that smoothing does have a very important effect. First of all, Gaussian smoothing, serve to reduce noise. Question 9.3 discusses this in detail, but the idea is that noise has the same high frequency as real features. The hope is that by using a Gaussian kernel, this noise will be reduced, and the Gabor filter can then respond only to actual features, which should be more robust to Gaussian smoothing. In practice, Gaussian smoothing serves two purposes in this case. First, it reduces the area of background that is detected as foreground, and also makes sure that some particular point in the foreground do not get treated as background. This is well represented in figure 15 if compared to figure 13.

(a) Robin-1 (b) Robin-2

Figure 15: Segmentation without Gaussian smoothing

## 5 Conclusion

To conclude, we review our most important discoveries. We got an insight on several denoising algorithms, on how they work, what makes them different and how we can tune them in order to perform at our expectations. We also learned about the importance of denoising of images in order to be used for more complex operations. Regarding edge detection, it seems that the Laplacian is a fairly robust edge detector, in all of its implementations. Segmentation using Gabor feature extractors also performs at a decent level, but more investigation should be done as to what choice of parameters have a grater and more positive influence on the segmentation results.

## References


[1] J. Movellan, *Tutorial on Gabor Filters*. [Online]. Available: http://mplab.ucsd.edu/tutorials/gabor.pdf

[2] K. Murthy, "Gabor filters : A practical overview," 2014. [Online]. Available: https://cvtuts.wordpress.com/2014/04/27/gabor-filters-a-practical-overview/

[3] L. Assirati, N. R. Silva, L. Berton, A. A. Lopes, and O. M. Bruno, "Performing edge detection by difference of gaussians using q-gaussian kernels," *Journal of Physics: Conference Series*, vol. 490, p. 012020, 2014.