Local image structures and morphology
Computer Vision, Lab Exercise 2
Andreas van Cranenburgh (0440949), Willem Bressers (5687063)
January 19, 2008

Abstract

This report discusses the use of the first and second order derivates for edge detection, and erosions and dilations for shape detection

1 Problem specification

Suppose a cookie factory checks it's cookies on a high speed conveyor belt by taking pictures of each cookie. A controller can't keep up with the speed so the images needs to be checked automatically. But how?

2 Theory and implementation

To solve the problem there are a few steps to take. First the edge of each cookie has to be detected. Next it's nessecary to thin the edges. And third the shape of the cookies need to be checked (for broken or missing parts).

2.1 Convolutions

A convolution is an operator that can be applied to an image using a convolution matrix. It is defined as integrating the product of two functions (f and a mirrored version of g). In discrete terms, for every pixel its neighborhood is multiplied with the convolution matrix, and the results are summed.

2.1.1 Gaussian kernel

The Gaussian kernel is an equation representing the canonical convolution matrix. Effectively it that blurs an image to a certain degree, depending on its scale. The scale is determined by the variable σ. For small values of σ, the Gausse curve (see figure 1) is taller and the curve is less broad.

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

This function returns a matrix of values scaled by σ. The dimensions of the grid are a multiplication of 2.5 times the σ The grid should always be at least the size of σ, but not too big either. The summation of this matrix should approach one, since the summation is a discrete approximation of integrating the Gauss function. With bigger σ, and thus bigger grids, the Gauss curve becomes very broad and the cutoff caused by approximating the integration becomes bigger.

So far, the Gaussian kernel has been calculated for two dimensions. By using the separability property of the convolution the same effect can be obtained by applying a one dimensional Gaussian kernel and then its transpose consecutively. The equation is very similar to the previous one:

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

The resulting vector makes convolutions much faster, since there are much less values to be multiplied and summed. Figure 2 shows a graph of the execution times of applying one and two dimensional Gaussian kernels on the same image, with a sigma ranging from 1 to 15.

2.1.2 Gaussian derivatives

It is also possible to approximate the first and second order derivative of an image by convoluting with the respective derivative of the Gaussian kernel. The first order derivative of an image f multiplied by the Gaussian kernel (with respect to x):

$$\frac{\delta G_\sigma}{\delta x} = -\frac{x}{\sigma^2} G_\sigma \tag{3}$$

This equation leads to a new image that looks rather like an embossed version of the original. Because it is the derivative, the actual colors have been replaced by gradients representing the change

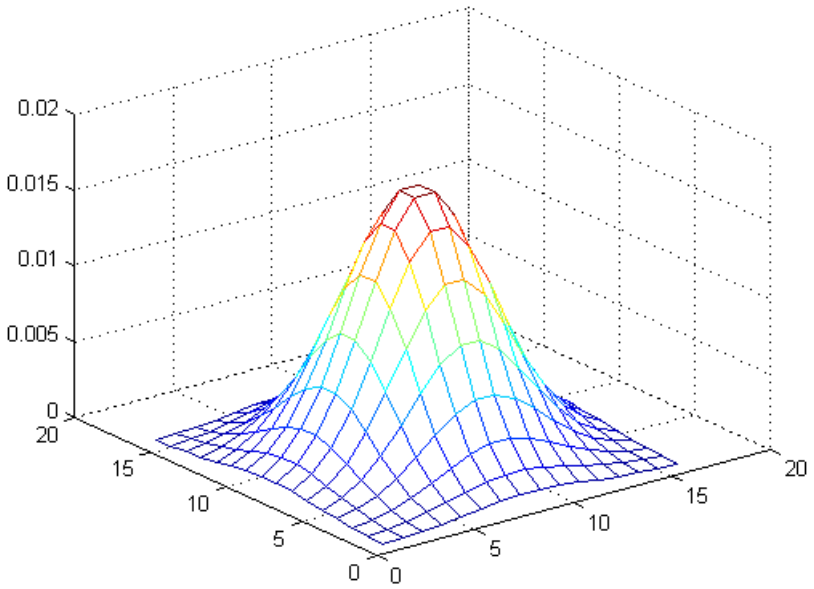

Figure 1: 3D plot of Gaussian kernel with $\sigma = 3$

in color in the x direction. The derivative can also be taken with respect to y, which will show vertical gradients. It is also possible to calculate the second derivative of the image:

$$\frac{\delta^2 G_\sigma}{\delta x^2} = (\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2})G_\sigma \tag{4}$$

By calculating the second derivative, the edges of the image become more salient. There are three kinds of second order derivatives, namely f_{xx}, f_{xy} and f_{yy}. Figure 3 shows the first and second order derivative in both directions.

2.1.3 Implementation of the Gaussion derivative

```
function g  = gD(f, sigma, xorder, yorder)
X = [-2.5*sigma:2.5*sigma];
% Zeroth, first and second gaussian derivatives:
gD = [ Gauss1(sigma); ...
       (-X ./ (sigma^2)) .* Gauss1(sigma); ...
       (-1 / (sigma^2)) .* ((1 - (X.^2) / (sigma^2))) .* Gauss1(sigma) ];
%Apply both x and y derivatives to image
%(note that gD of y has to be transposed)
g = imfilter(f, gD(xorder+1,:), 'conv', 'replicate');
g = imfilter(g, gD(yorder+1,:)', 'conv', 'replicate');
```

2.2 Analytic and Gaussian derivative

The second order f_{xy} Gaussian derivative is a powerful tool to detect edges in an image. The following example demonstrates the analytic and Gaussian (numerical) derivative of a function.

2.2.1 example

Asume an image described by the following function:

$$f(x, y) = A\sin(Vx) + B\cos(Wy) \tag{5}$$

Before the seconde order derivative can be calculated, it's necessary to calculate the first order derivatives f_x, f_y of the function f:

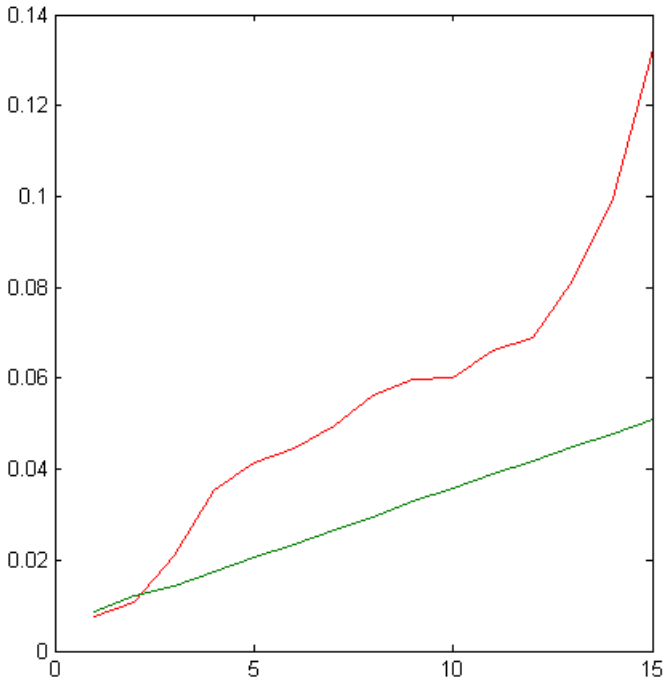

Figure 2: Difference in performance between applying one and two dimensonial Gaussian kernels. The complexity thus appears to be $O(\sigma)$ and $O(\sigma^2)$, respectively.

$$f_x = (V) * (A\cos(Vx)) \tag{6}$$

$$f_y = (W) * (B - \sin(Wy)) \tag{7}$$

Now it's possible to calculate the second order derivatives f_{xx}, f_{yy}, f_{xy} of the function f:

$$f_{xx} = (V^2)(A - \sin(Vx)) \tag{8}$$

$$f_{yy} = (W^2) * (B\cos(Wy)) \tag{9}$$

$$f_{xy} = A\cos(V) + B\cos(W, y) - B\sin(W) \tag{10}$$

After differentiating this function analytically, it is possible to compare that with the Gaussian derivative of a discretized version of $f(x, y)$. Figure 4 shows that they are very similar. Except that for the numerical approximation, the vectors had to be translated by 105 pixels in both the x and y direction. The reason for this remains elusive to us. The figure has been superimposed with a grid of vectors showing the magnitude and direction of change on the sampled locations (every 10 pixels for both x and y)

2.3 Canny edge detector

The Canny edge detector uses the first order Gaussian derivative to find the edges of an image. First it takes the gradient norm of the image, which represents the magnitude of biggest change for every pixel, indepent of the coordinate system, using this formula:

$$f_w = \sqrt{f_x^2 + f_y^2} \tag{11}$$

But this is still only the *amount* of change, we also need the direction (angle):

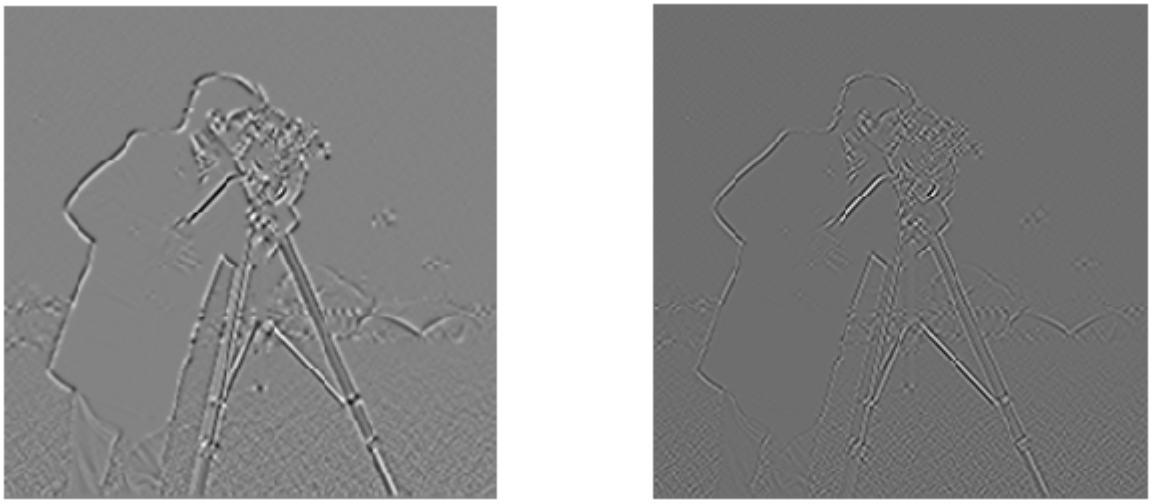

Figure 3: On the left the 1-jet, on the right the 2-jet (second derivative in both x and y direction).

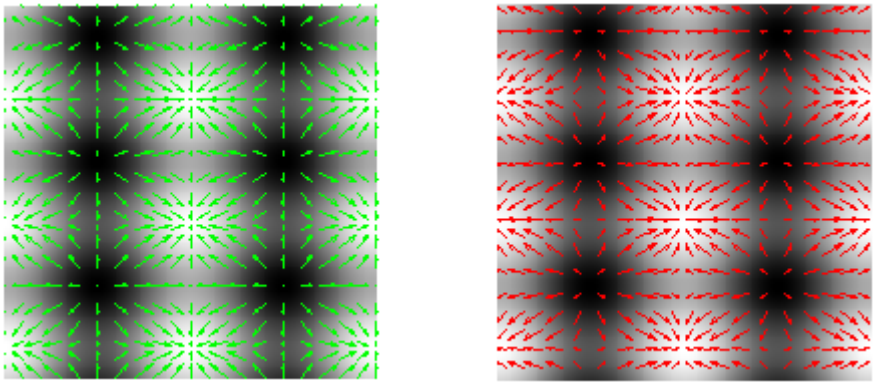

Figure 4: the gradient vectors, analytic (green) and numeric (red)

$$f_\theta = \arctan \frac{f_x}{f_y} \tag{12}$$

Because the algorithm depends on the Gaussian derivative, σ can be specified. This parameter depends the scale or level of detail to be considered. With $\sigma = 1$ many edges will be detected (likely too many), increasing it will gradually do away with less pronounced edges.

The next step is filtering out non-maxima in the gradient norm (also called 'thinning'). Armed with the angles of greatest change, we can search for pixels without smaller neighbors in the appropriate direction. Pixels that fail the test will be black, edges will be white. Since we are dealing with discrete images, the angles have to be rounded to corespond to one of the four possible directions (given eight neighbor pixels). Figure 5 shows the result of the Canny edge detection on an example image. It should be noted that our implementation does not use *hysterisis*. This amounts to using two additional parameters specifying a minimum value for an edge to begin, and for an edge to stop respectively.

2.4 Morphology

"Mathematical Morphology" is a tool to detect objects in an image. For example assume an image of a plane that that flies in clear air. With Morphology it is possible to detect the blue sky and filter it out. It is harder to detect the plane because it would have shadows and different colors.

Figure 5: Canny edge detection with $\sigma = 1$ using thinning.

2.4.1 Isodata

Before anything can be detected it is necessary to find the colors of the object. Given an appropriate background, it is possible to partition the pixel values of an image in background and foreground, respectively. This can be done by looking at the histogram of the image and finding an appropriate treshold. This can be done non-linearily (iteratively). An initial guess might be the mean of all pixel values. Then at each step the mean of all pixels below this threshold and those above is taken, until the resulting threshold reaches a plateau, or until a fixed number of iterations have been performed. See figure 6.

Consequently, each pixel value in the image is compared with the value of the treshold. This results in a binary version ("mask"). See figure 7

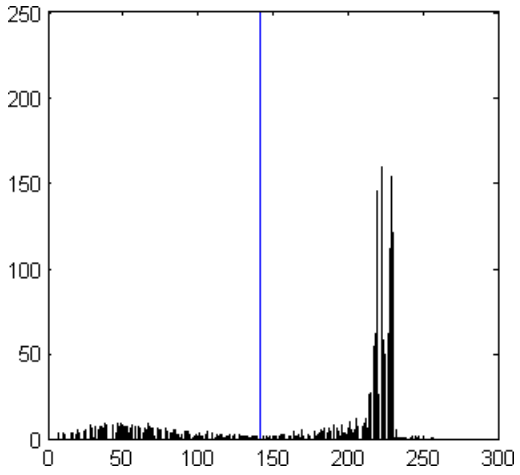

Figure 6: Histogram of an image (see figure 7). The blue line represents the isodata threshold

2.4.2 Erosion

Put simply, erosion 'eats away' the edges of white objects (with a certain structuring element of which the size and the shape are significant).

As can be seen on the example photo (see figure 7), eroding effectively erases the inscriptions on the coins. With a square of size 3 some dots remained, but with size 15 the coins consisted only of white circles. These circles become increasingly bigger as the size of the structure element increases.

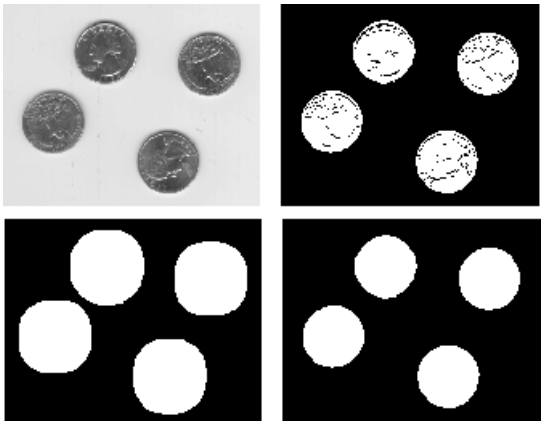

Figure 7: In respective order: the original, thresholded, eroded and dilated (both with a square of 15×15)

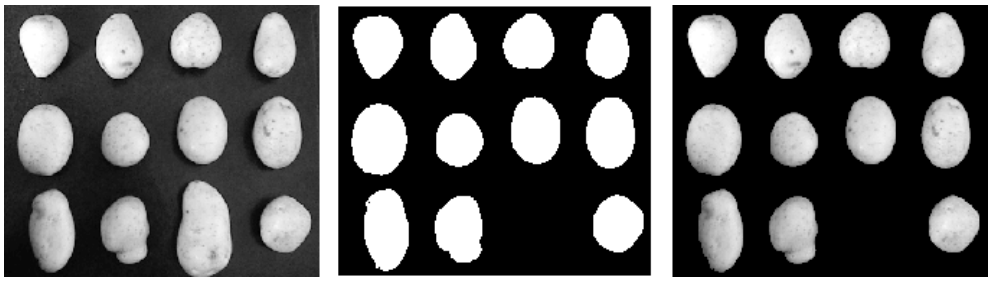

Figure 8: In respective order: the original, mask (threshold and border fill applied) and the result

When a structure element of size 30 is used, the coins can become so big as to touch each other, in effect connecting them. At size 60 the coins form one big blob.

2.4.3 Dilation

Dilation is in a certain sense the inverse of erosion, except that erosion throws away information that can not be regained. White objects will grow in size, according to the structuring element used.

Imagine a structure element *se* of size 7×7 filled with zeroes except in the upper-left corner. A dilation with such a structuring element would translate white objects by minus one pixel in both the x and y direction.

In the example image (figure 7) the coins have been restored to their original size. When the structure element is increased, dilating and eroding with the same structure element does not change the size of the coins, except that connectivity remains invariant. In practice this means that objects will feature a sort of 'leash' to the corner or to other objects, if they were connected before the dilation. Information that has been removed by a previous erosion will of course not reappear by dilating it again.

2.4.4 Border objects

When working with shapes on images, objects that touch the border can present a problem as their shape is incomplete. Thus it is useful to be able to automatically clear the borders of an image.

The leftmost photo in figure 8 has been filtered with a static treshold of 0.3 (using the isodata threshold removed parts of some of the potatoes). To remove some further noise, an opening (erosion followed by dilation) has been applied, with a structure element of size 1. This removes all spots on the potatoes, and makes the background uniform, as can been seen in the image in the middle. After this, it is trivial to apply this mask to the original, by multiplication (only the white parts of the mask influence the resulting image), as can be seen on the image on the right.

Figure 9: In respective order: the original, the marker (obtained by erosion), and the result (by reconstruction)

2.4.5 Selection on size

Erosions can also be applied to detect objects that adhere to certain shape constraints. We have applied this idea to a picture of two cookies, one of which is broken (see figure 9). After some experimenting with different parameters[1], we figured out a way to remove the broken cookie entirely.

First a threshold is applied by isodata. Then a dilation with a disk of size 5, to remove some noise. Then the crucial part, an erosion with a structure element in the shape of a disk size 65. This removes the broken cookie completely, and leaves a small dot in place of the other. In order to regain the original shape of this cookie, a reconstruction is applied with the result of the erosion as marker and the thresholded original as mask. Finally, the result of this can be multiplied with the original image to obtain the original pixel values of the cookie, as shown in the image on the right.

Before attempting to reconstruct the image we used a dilation with the same structuring element as the erosion, but this resulted in a cookie with the shape of a hexagon.

2.4.6 Openings and closings

Before the mask can be use to detect objects it is expedient to remove any binary noise. This can be done with a combination of dilation and erosion:

Closing:

$$(A \oplus S) \ominus S \tag{13}$$

Opening:

$$(A \ominus S) \oplus S \tag{14}$$

Where A is the mask and S is a "marker". The function Dilation first add's the marker at every binairy one. Next it substracts the marker from the mask. The function Erosion first substracts the marker followed by adding the same marker. By using first the Dilate function and next the Erosion function most of the noise is removed without loosing border edges.

3 Optimizations and implementations

4 Usage

5 Conclusion

6 References

[1] Fortunately this experimenting only has to be done once, as long as the cookies stay the same size and the camera position is fixed in relation with the cookies, as these operations are defined to be translation and rotation invariant.