Assignment 6

Birk Nøhr Dissing
Jakob Damgaard Olsen
Ian Pascal Møller

March 20, 2023

1. Fixed scale feature detectors

1.1

Canny features was found on the image "hand.tiff". The original image can be seen in figure 1. The Canny features was found with different settings, the resulting images is shown in figure 2. Here it can be seen that when σ is increased the lines in the image becomes smoother and sharp features disappear as the image is more blurred. The values for low and high threshold determines the number of points present in the output. When they are increased the number of points decreases.

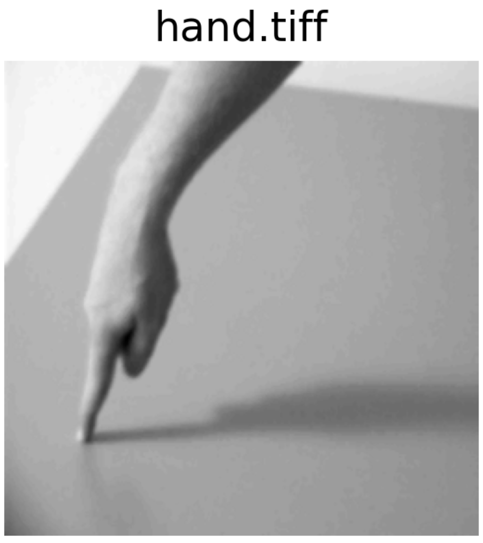

Figure 1: The original image of "hand.tiff".

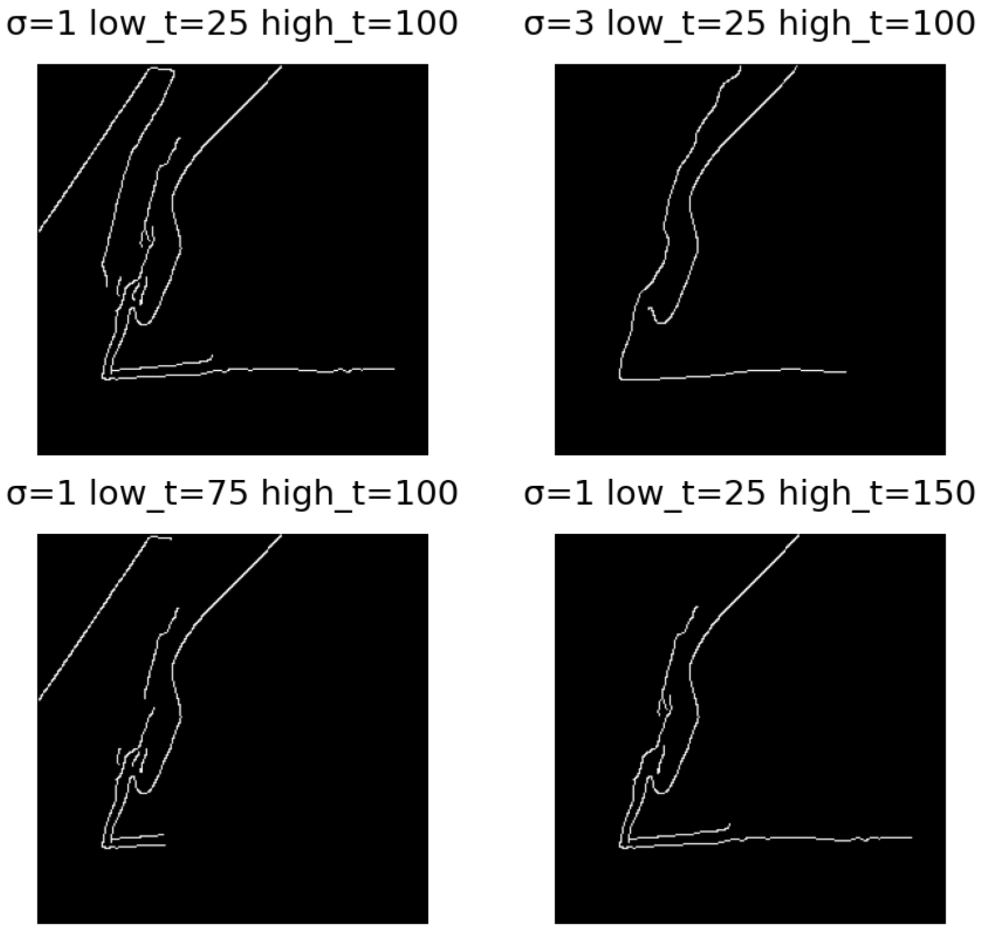

Figure 2: The Canny feature of "hand.tiff" with different setting for σ, low threshold(low_t), and high threshold(high_t).

1.2

The Harris corner response image of the "modelhouses.png" was created using both the "k" and "eps" methods. Various values of σ were used for each image, while various values for k and eps were used was used for their respective methods. The original "modelhouses.png" image can be seen in figure 3. The response image created with the "k" method can be seen in figure 4. Here it can be seen that response image becomes more blurred when σ is increased. This also makes it so that larger features are more visible like roofs and windows. Increasing the k value reduces the number of high value pixels in the response image.

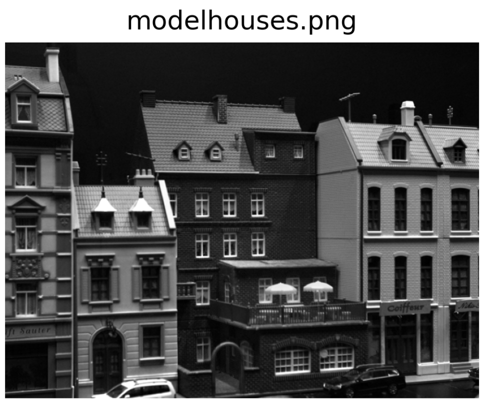

Figure 3: The original image of "modelhouses.png".

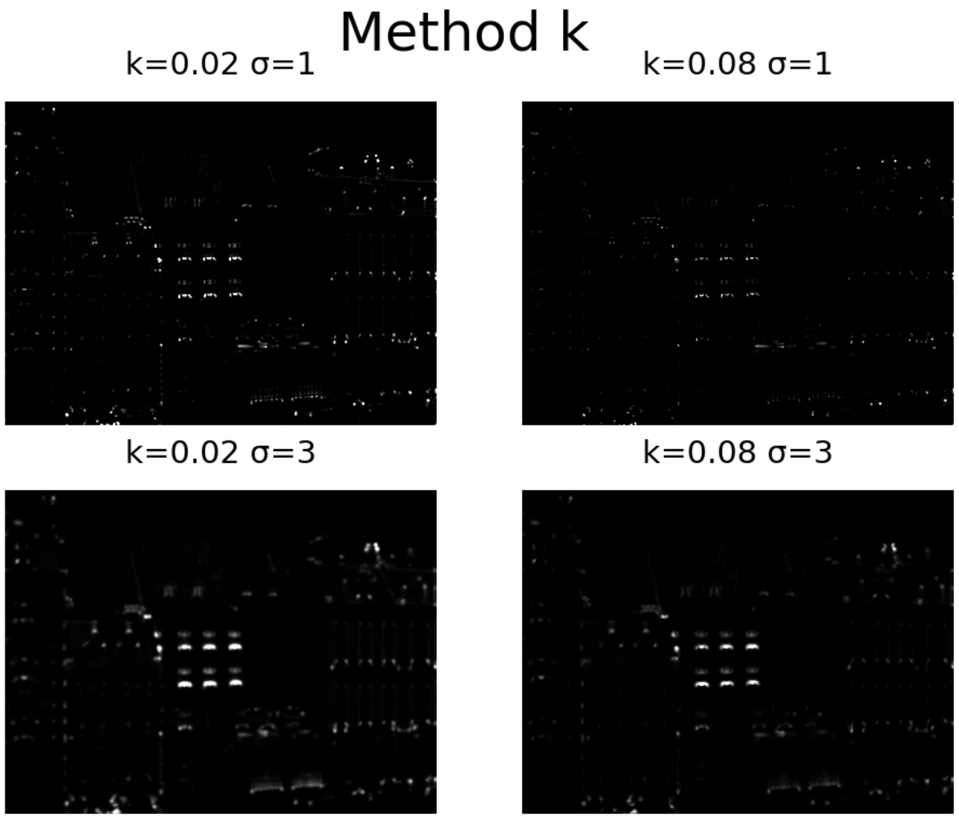

Figure 4: Harris corner response images of the "modelhouses.png" were created using the "k" method with various values for k and σ.

The response images from the "eps" method can be seen in figure 5. Here it can again be seen that a higher σ value blurs the images but makes larger features more visible. A higher eps value results in fewer high intensity points in the response image.

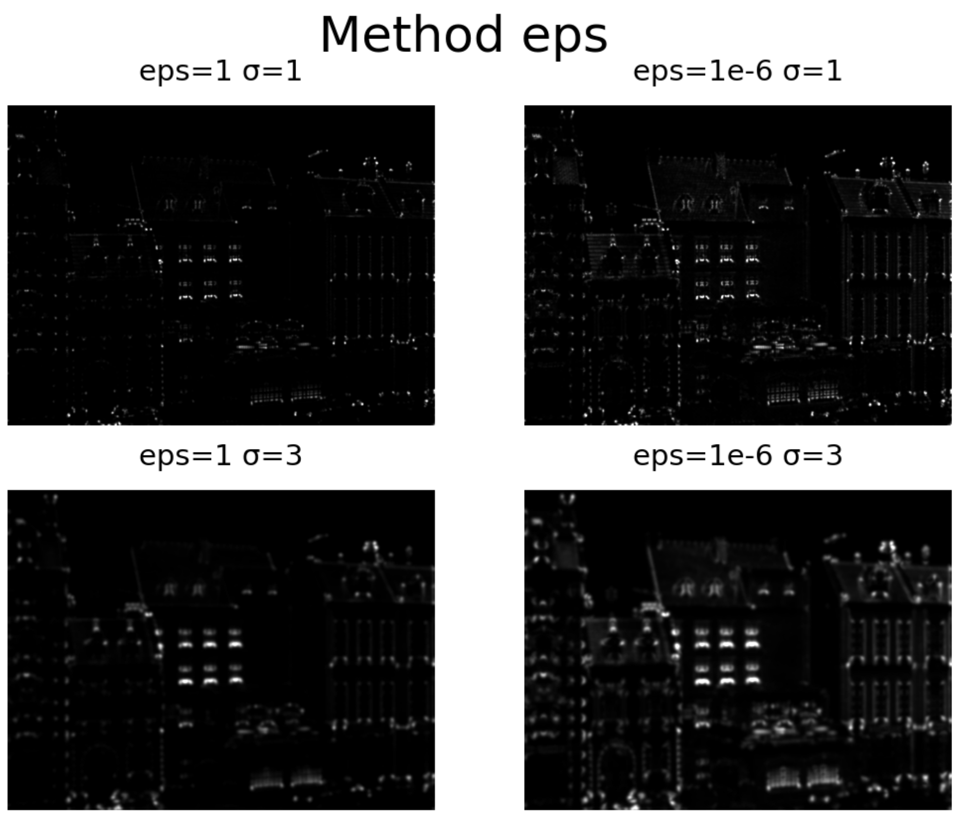

Figure 5: Harris corner response images of the "modelhouses.png" were created using the "eps" method with various values for eps and σ.

1.3

The following function finds the N strongest Harris corner points in an image using either the "k" or "eps" method.

```python
def find_corners(image, N_corners, method="k", k=0.05, eps=1e-06,
   sigma=1):
    image_corner = corner_harris(image, method, k, eps, sigma)
    return corner_peaks(image_corner, num_peaks=N_corners)
```

The function was used to find the 250 strongest Harris corners in the "modelhouses.png". This was done using the "k" method and the parameters $\sigma = 1$ and $k = 0.05$. The corner points found can be seen in figure 6.

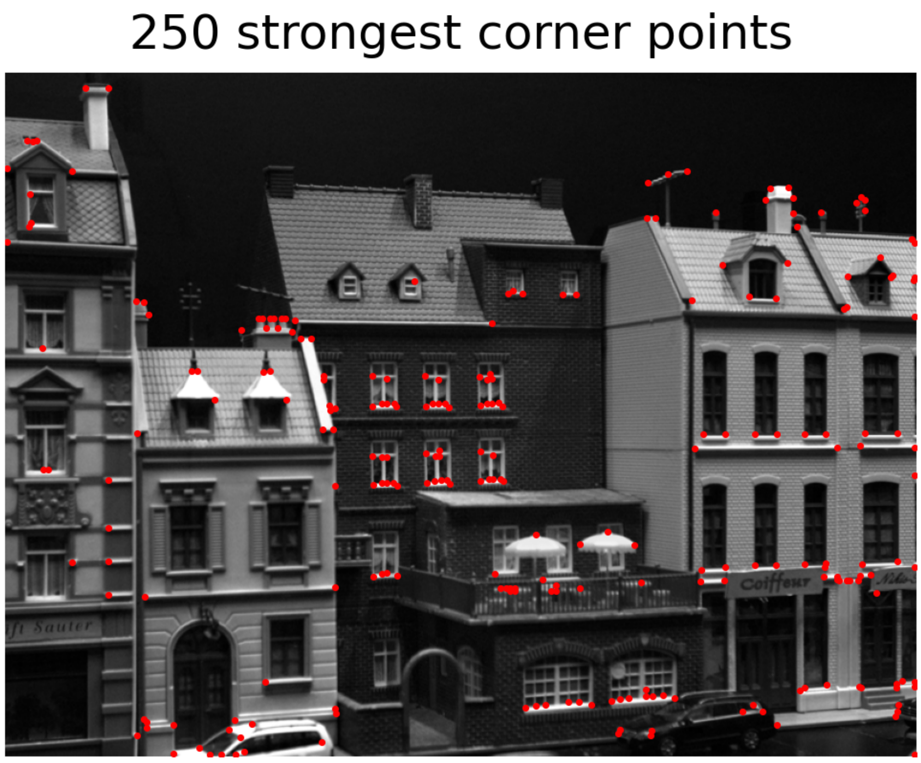

Figure 6: The 250 strongest corner points in the "modelhouses.png" image were found using the "k" method with $\sigma = 1$ and $k = 0.05$.

2. Feature detection and image transforms

The parameters used for the "find_corners" function described in exercise 1.3 was the following : **N_corners = 50, k = 0.05, sigma = 7**. Using these parameters we were able to detect the corners of the paper, but it also detected the corners of the image. We therefore removed the 4 outer most corner first, before labeling the corners of the paper. After labeling the corners of the paper two lines were created and the angle between the point forming these two lines were calculated. An average of the two angels was then used to find the needed rotation which turned out to be **-90.5465 degrees**. The way this was implemented in code was the following.

```python
def angle_between_points(p1,p2):
    delta_x = p2[1] - p1[1]
    delta_y = p2[0] - p1[0]
    return np.arctan2(delta_y, delta_x)
A  = imread("textlabel_gray_small.png")
corners = find_corners(A,50,k=0.05,sigma=7)
top_left = corners[np.argmin(np.sum(corners, axis=1))]
top_right = corners[np.argmin(np.diff(corners, axis=1))]
bottom_right = corners[np.argmax(np.sum(corners, axis=1))]
bottom_left = corners[np.argmax(np.diff(corners, axis=1))]
indices = []
for corner in [top_left, top_right, bottom_right, bottom_left]:
    index = np.where((corners == corner).all(axis=1))[0][0]
    indices.append(index)
corners = np.delete(corners, indices, axis=0)
top_left = corners[np.argmin(np.sum(corners, axis=1))]
bottom_left = corners[np.argmin(np.diff(corners, axis=1))]
bottom_right = corners[np.argmax(np.sum(corners, axis=1))]
top_right = corners[np.argmax(np.diff(corners, axis=1))]
Ang1 = angle_between_points(bottom_left,top_left)
Ang2 = angle_between_points(bottom_right,top_right)
AngAvg = (Ang1 + Ang2) / 2
AngDegr = np.rad2deg(AngAvg)
rotated_image = rotate(A,(AngDegr),resize=True)
```

The resulting image from this process can be seen in figure 7. Here we see the points found from the ”find_corners” function, with the points used to create lines marked as red dots and green dots respectively. We see that the rotation has been a success and the text is horizontal making it easier to read it.

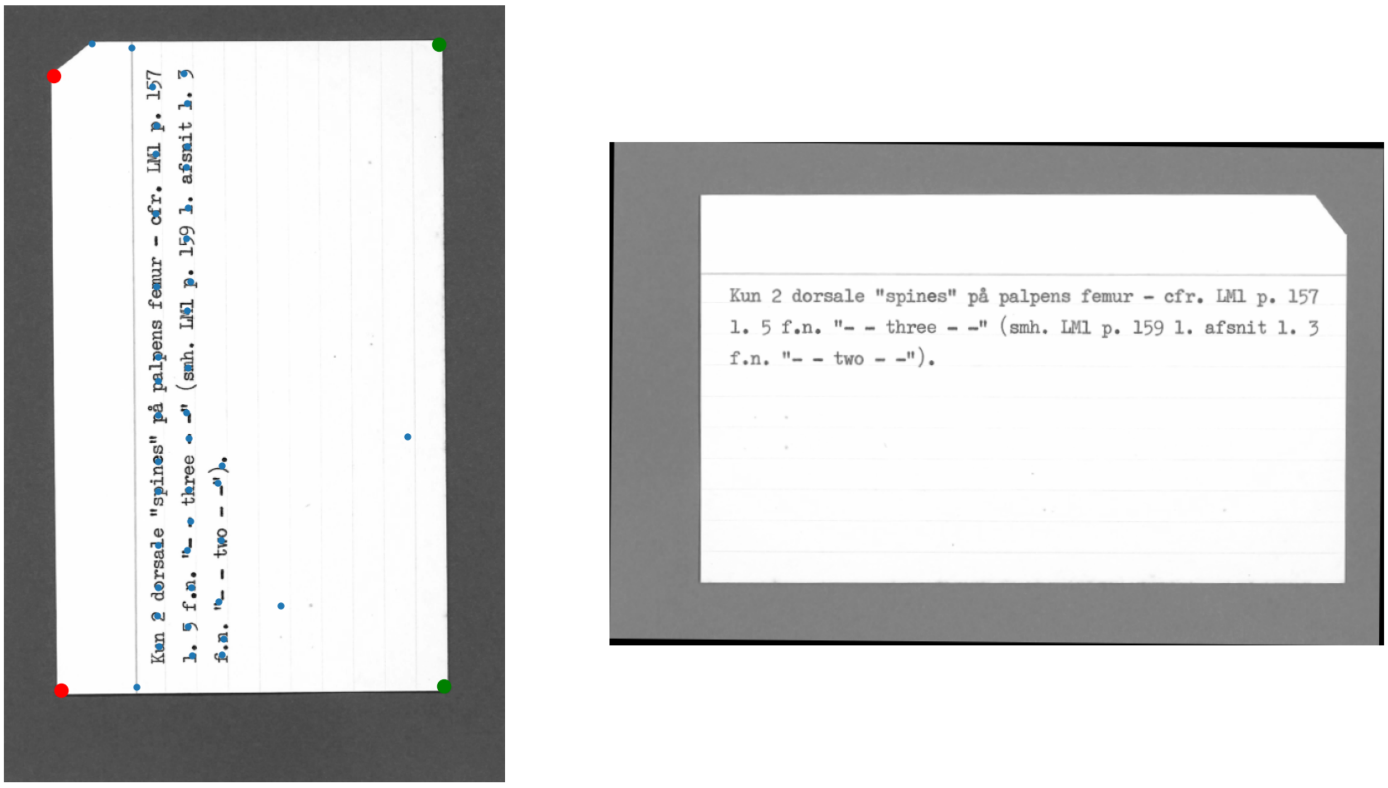
Kun 2 dorsale "spines" på palpens femur - cfr. LM1 p. 157
l. 5 f.n. "- - three - -" (smh. LM1 p. 159 l. afsnit l. 3
f.n. "- - two - -").

Figure 7: Left image shows the original image with the results of finding the corners of it. The points used to create two lines are marked with the color red and green respectively. The right image shows the original image rotated by -90.5465 degrees.

3. Scale-space blob detector

3.1

The function seen in the code snippet below was made to generate the discrete images we need. It takes in a size parameter that determines the length and width of the resulting image, and then a func parameter, which is the function to generate the actual pixel values at each point, which in our case here just is a separate implementation of equation 3 in the assignment text. The last parameter in then in this case σ, the width. The image can be generated simply by iterating over each element in a matrix and calculating eq. 3 for each of those, using the matrix indices as the arguments. We then calculate the values for the matrix symmetrically around the point (0, 0) and then afterwards shift the matrix by half its size in each direction. This results in a Gaussian distribution that is also centered.

```python
def generate_image(size, func, func_args):
    img = np.zeros((size, size))

    half_size = int((size / 2))
    rng = range(-half_size, half_size+1)
    for i in rng:
        for j in rng:
            img[i,j] = func(i, j, func_args)

    img = np.roll(img, shift = (half_size, half_size), axis = (0,
       1))

    return img
```

The collection of images in fig. 8 shows various results: starting from the left, we have a generated image using the implemented function above, with width (sigma) equal to 1. Then, we have the same, but with width (tau) equal to 2. The middle image shows the convolution of these two images. The fourth image shows the image obtained when generating a Gaussian blob using the function mentioned before, with width equal to the square root of the sum of squares of the two individual Gaussian distribution's widths. This should be equal to the third image, since by eq. 1 in the assignment text we know that this should be an equivalent expression for the convolution of two Gaussians. The final image to the right shows the (absolute) difference between the previous image, and the convolution (third) image. From this, we can see that there is a very small difference, and only in certain points. Note the differing scale: the values in the images are at most about 0.15, while in the difference image the values are at most about $1.5 \cdot 10^{-8}$, indicating that there is very little difference, however there is clearly some pattern to it. The difference is quite small though and is most likely due to floating point precision.

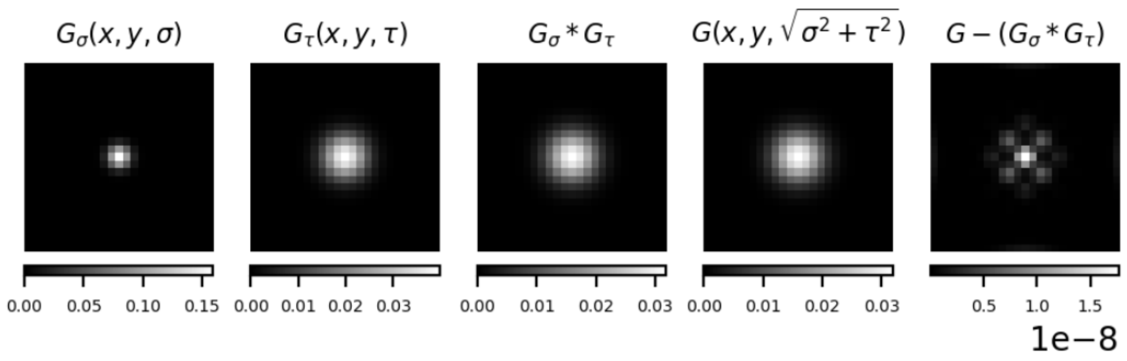

Figure 8: Series of images showing the process and result of generating Gaussian-blob images. See text for explanation.

3.2

We need to prove that a Gaussian convoluted with another Gaussian is equal to a Gaussian.

$$g(x,y,\sigma) * g(x,y,\tau) = g(x,y,\sqrt{\sigma^2+\tau^2}) \tag{1}$$

This is done by utilising the the convolution theorem for Fourier transformation.

$$\mathcal{F}(g(x,y,\sigma) * g(x,y,\tau)) = \mathcal{F}(g(x,y,\sigma)) \cdot \mathcal{F}(g(x,y,\tau)) \tag{2}$$

where $\mathcal{F}$ is the Fourier transformation of a function. We can then rewrite equation 1 by taking the Fourier transform of both sides.

$$\mathcal{F}(g(x,y,\sigma)) \cdot \mathcal{F}(g(x,y,\tau)) = \mathcal{F}(g(x,y,\sqrt{\sigma^2+\tau^2})) \tag{3}$$

The 2 dimensional Fourier transformation of a Gaussian is found.

$$\mathcal{F}(g(x,y,\sigma)) = \frac{1}{2\pi\cdot\sigma^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}}dy \quad dx \tag{4}$$

$$= \frac{1}{2\pi\cdot\sigma^2}\int_{-\infty}^{\infty}\mathrm{e}^{-\frac{y^2}{2\sigma^2}}dy\int_{-\infty}^{\infty}\mathrm{e}^{-\frac{x^2}{2\sigma^2}}dx = \frac{1}{2\pi\cdot\sigma^2}\cdot\sqrt{2\pi\cdot\sigma^2}\cdot\sqrt{2\pi\cdot\sigma^2}\cdot\mathrm{e}^{-2\pi^2v^2\sigma^2}$$

$$= \mathrm{e}^{-2\pi^2\left(u^2+v^2\right)\sigma^2}$$

The first part of equation 3 is evaluated.

$$\mathcal{F}(g(x,y,\sigma)) \cdot \mathcal{F}(g(x,y,\tau)) = \mathrm{e}^{-2\pi^2\left(u^2+v^2\right)\sigma^2}\cdot\mathrm{e}^{-2\pi^2\left(u^2+v^2\right)\tau^2} \tag{5}$$

$$= \mathrm{e}^{-2\pi^2\left(u^2+v^2\right)\left(\sigma^2+\tau^2\right)}$$

The second part of equation 3 is evaluated.

$$\mathcal{F}(g(x,y,\sqrt{\sigma^2+\tau^2})) = \mathrm{e}^{-2\pi^2\left(u^2+v^2\right)\left(\sqrt{\sigma^2+\tau^2}\right)^2} \tag{6}$$

$$= \mathrm{e}^{-2\pi^2\left(u^2+v^2\right)\left(\sigma^2+\tau^2\right)}$$

Equation 1 is then fulfilled.

$$g(x,y,\sigma) * g(x,y,\tau) = g(x,y,\sqrt{\sigma^2+\tau^2}) \tag{7}$$

$$\Leftrightarrow \mathcal{F}(g(x,y,\sigma)) \cdot \mathcal{F}(g(x,y,\tau)) = \mathcal{F}(g(x,y,\sqrt{\sigma^2+\tau^2}))$$

$$\Leftrightarrow \mathrm{e}^{-2\pi^2\left(u^2+v^2\right)\left(\sigma^2+\tau^2\right)} = \mathrm{e}^{-2\pi^2\left(u^2+v^2\right)\left(\sigma^2+\tau^2\right)}$$

3.3

i.

The image $I(x, y, \tau)$ is the convolution of two Gaussians

$$I(x, y, \tau) = g(x, y, \sigma) * g(x, y, \tau) = g(x, y, \sqrt{\sigma^2 + \tau^2}) \tag{8}$$

We need to find the closed form expression of $H(x, y, \tau)$.

$$H(x, y, \tau) = I_{xx}(x, y, \tau) + I_{yy}(x, y, \tau) \tag{9}$$

Where

$$I_{xx}(x, y, \tau) = \tau^2 \cdot \frac{\partial^2 I(x, y, \tau)}{\partial x^2} \quad I_{yy}(x, y, \tau) = \tau^2 \cdot \frac{\partial^2 I(x, y, \tau)}{\partial y^2} \tag{10}$$

$I_{xx}(x, y, \tau)$ is found.

$$I_{xx}(x, y, \tau) = \tau^2 \cdot \frac{\partial^2 I(x, y, \tau)}{\partial x^2} = \tau^2 \cdot \frac{\partial}{\partial x}\left(\frac{\partial I(x, y, \tau)}{\partial x}\right) \tag{11}$$

$$= \tau^2 \cdot \frac{\partial}{\partial x}\left(-\frac{\tau^2 x \mathrm{e}^{-\frac{x^2+y^2}{2\sqrt{\sigma^2+\tau^2}}}}{2\pi\left(\sigma^2+\tau^2\right)}\right)$$

$$= \tau^2 \left(-\frac{\mathrm{e}^{-\frac{x^2+y^2}{2\sqrt{\sigma^2+\tau^2}}}}{2\pi\left(\sigma^2+\tau^2\right)} + \frac{x^2 \mathrm{e}^{-\frac{x^2+y^2}{2\sqrt{\sigma^2+\tau^2}}}}{2\pi\left(\sigma^2+\tau^2\right)^{\frac{3}{2}}}\right)$$

The same can be done for $I_{yy}(x, y, \tau)$ yielding the following.

$$I_{yy}(x, y, \tau) = \tau^2 \left(-\frac{\mathrm{e}^{-\frac{x^2+y^2}{2\sqrt{\sigma^2+\tau^2}}}}{2\pi\left(\sigma^2+\tau^2\right)} + \frac{y^2 \mathrm{e}^{-\frac{x^2+y^2}{2\sqrt{\sigma^2+\tau^2}}}}{2\pi\left(\sigma^2+\tau^2\right)^{\frac{3}{2}}}\right) \tag{12}$$

The expression for $H(x, y, \tau)$ can then be reduced.

$$H(x, y, \tau) = I_{xx}(x, y, \tau) + I_{yy}(x, y, \tau) \tag{13}$$

$$= \tau^2 \left(-\frac{\mathrm{e}^{-\frac{x^2+y^2}{2\sqrt{\sigma^2+\tau^2}}}}{2\pi\left(\sigma^2+\tau^2\right)} + \frac{x^2 \mathrm{e}^{-\frac{x^2+y^2}{2\sqrt{\sigma^2+\tau^2}}}}{2\pi\left(\sigma^2+\tau^2\right)^{\frac{3}{2}}}\right) + \tau^2 \left(-\frac{\mathrm{e}^{-\frac{x^2+y^2}{2\sqrt{\sigma^2+\tau^2}}}}{2\pi\left(\sigma^2+\tau^2\right)} + \frac{y^2 \mathrm{e}^{-\frac{x^2+y^2}{2\sqrt{\sigma^2+\tau^2}}}}{2\pi\left(\sigma^2+\tau^2\right)^{\frac{3}{2}}}\right)$$

$$= \frac{\tau^2}{2\pi\left(\sigma^2+\tau^2\right)} \mathrm{e}^{-\frac{x^2+y^2}{2\sqrt{\sigma^2+\tau^2}}} \cdot \left(\frac{y^2+x^2}{\sqrt{\left(\sigma^2+\tau^2\right)}} - 2\right)$$

Which yields the closed form expression for $H(x, y, \tau)$.

$$H(x, y, \tau) = \frac{\tau^2}{2\pi\left(\sigma^2+\tau^2\right)} \mathrm{e}^{-\frac{x^2+y^2}{2\sqrt{\sigma^2+\tau^2}}} \cdot \left(\frac{y^2+x^2}{\sqrt{\left(\sigma^2+\tau^2\right)}} - 2\right) \tag{14}$$

ii.

We want to evaluate the extremas for $H(0,0,\tau)$.

$$H(0,0,\tau) = \frac{\tau^2}{2\pi\left(\sigma^2+\tau^2\right)} \mathrm{e}^{-\frac{0^2+0^2}{2\sqrt{\sigma^2+\tau^2}}} \cdot \left(\frac{0^2+0^2}{\sqrt{\left(\sigma^2+\tau^2\right)}} - 2\right) \tag{15}$$

$$= \frac{-\tau^2}{\pi\left(2^2+\tau^2\right)}$$

In order to find extremal points for $H(0,0,\tau)$ we look for τ such that

$$\frac{\mathrm{d}H(0,0,\tau)}{\mathrm{d}x} = -\frac{2\tau}{\pi\left(\tau^2+4\right)} + \frac{2\tau^3}{\pi\left(\tau^2+4\right)^2} = 0 \tag{16}$$

Using Maple to solve this yields $\tau = 0$

$$\frac{\mathrm{d}H(0,0,0)}{\mathrm{d}x} = 0 \tag{17}$$

In order to see if the point is a maxima or minima the following is evaluated.

$$\frac{\mathrm{d}^2H(0,0,0)}{\mathrm{d}x^2} = -\frac{2}{\pi\left(0^2+4\right)} + \frac{10\cdot 0^2}{\pi\left(0^2+4\right)^2} - \frac{8\cdot 0^4}{\pi\left(0^2+4\right)^3} \tag{18}$$

$$= -\frac{1}{2\pi}$$

As $\frac{\mathrm{d}^2H(0,0,0)}{\mathrm{d}x^2}$ is negative the extremal point at $\tau = 0$ is a maxima.

iii.

Equation 15 was evaluated for 1000 evenly spaced values of τ between -10 and 10 using python. The result is plotted in figure 9. Here it can be seen the there is an extremal point at $\tau = 0$ which is a maxima. This confirms the solution which was arrived at analytically in problem 3.3.ii.

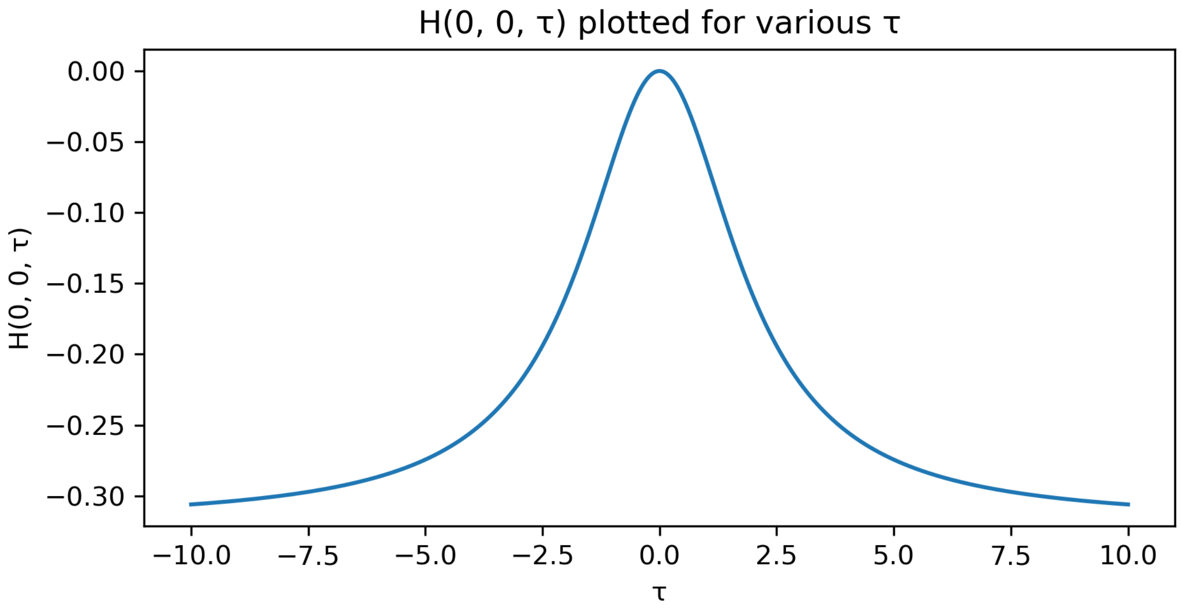

Figure 9: $H(0,0,\tau)$ plotted as a function of τ.

3.4

To implement blob detection with scale selection, we calculated the Laplacian of our image convoluted with a Gaussian. This result was then scaled based on the value of σ used. The code used to do blob detection for an image given a range of σ is the following, where first the Laplacian of the image convolved with a Gaussian is found. Then this value is used to calculate the scale space by multiplying with σ^2. In the end the 150 points are found from the maxima and the minima of the scale space.

```python
def blob_detection(img,scales):
    L = np.zeros((img.shape[0], img.shape[1], len(scales)))
    for i, s in enumerate(scales):
        L[:,:,i] = sm.filters.laplace(sm.filters.gaussian(img,
           sigma=s))
    scaleSpace = np.zeros((img.shape[0], img.shape[1], len(scales)
       ))
    for i, s in enumerate(scales):
        scaleSpace[:,:,i] = s**2 *(L[:,:,i])
    maxCoords = sm.feature.peak_local_max(scaleSpace, num_peaks
       =150)
    minCoords = sm.feature.peak_local_max(scaleSpace.max()-
       scaleSpace,num_peaks=150)
    return maxCoords, minCoords
```

The result of using this function on the image "sunflower.tiff" with $\sigma = [0, 30]$ can be seen in figure 10. Here that the minima (blue) of the function gives us the sunflower heads, while the maxima (red) gives us the tip of the petals. The circles indicate the value of σ where we can see that larger values of sigma are in the front of the image while lower values are in the back. For the minima the lowest value of σ was 0.606 while the largest was 8.48. For the maxima the lowest value was 1.21 while the largest was 25.15.

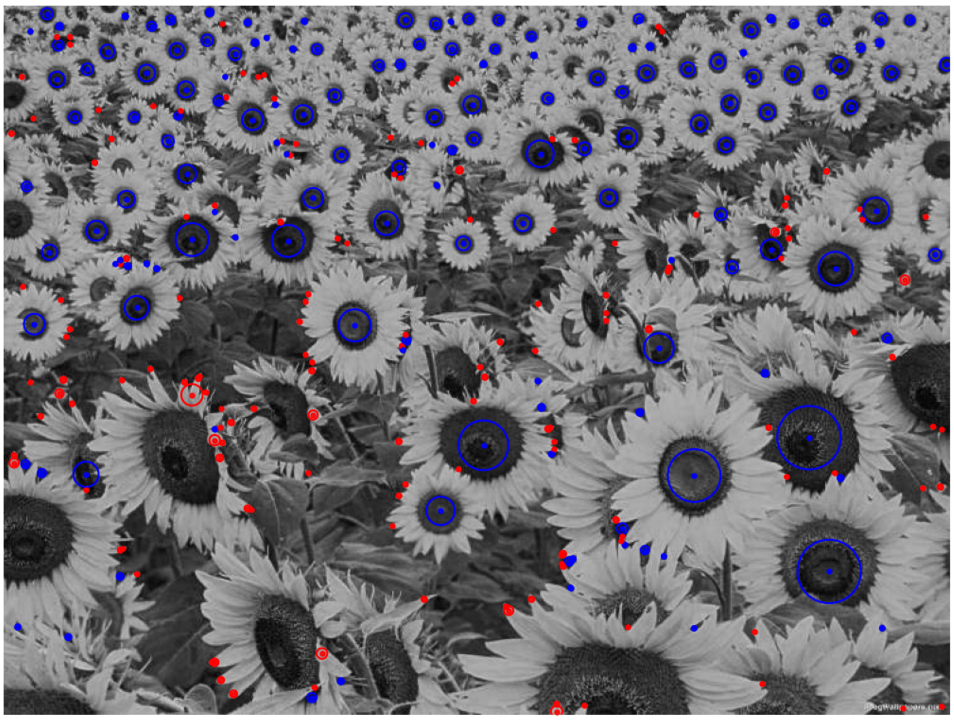

Figure 10: Result of using blob detection with scale selection for sigma in the range $\sigma = [0, 30]$. The blue dots indicate the minima while the red dots indicate the maxima. The circles indicate the sigma for each point.