Fundamentals of Computer Vision

Assignment #1

Name: Asa Borzabadi Farahani

(Ph.D. Student in Neurosciences)

Student ID: 761002026

TABLE OF CONTENTS

Question Number One - Marr-Hildreth 5

Part a 5

Answer – Part A 5

Part B 7

Answer – Part B 7

Part C 10

Answer – Part C 10

Question Number Two - Edge detection Using Oriented Operators 11

Part a 11

answer – Part A 11

Part B 13

answer – Part B 13

PART C 14

answer – Part C 14

PART D 16

answer – Part D 16

PART E 22

answer – Part E 22

Question Number Three - Line finding using RANSAC 26

Part A 26

Answer – Part A 26

Part B 30

Answer – Part B 30

Refrences 32

TABLE OF CONTENTS - FIGURES

Figure 1 Suppose this is your intensity profile near the edge, the red line is the direction of one of the second derivatives and the black line is the direction of another second derivative, in this case the black direction is the one known as the best direction, since when looking at the profile of second derivatives, this directional second derivative has the strongest zero-crossing which means in this case the slope of zero-crossing is the largest among other directional derivatives. 7

Figure 2 This is the profile of the second directional derivative in the red direction. Here, the slope of zero-crossing is not that sharp....8

Figure 3 This is the profile of the second directional derivative in the black direction. Here, the slope of zero-crossing is sharp....8

Figure 4 Block diagram of the Marr-Hildreth algorithm [2]....9

Figure 5 Example of a situation where linear variation constraint is not met (a corner)....9

Figure 6 Example of a case when Marr-Hildreth edge detector might collapse [3] 10

Figure 7 Gaussian filter is shown in red and its second derivative is illustrated in blue. Size of array for the second derivative of Gaussian is equal to $(\boldsymbol{2\,\sigma + 1})$ in this case. 12

Figure 8 Gaussian filter is shown in red and its second derivative is illustrated in blue. Size of array for the second derivative of Gaussian is equal to $(\boldsymbol{6\,\sigma + 1})$ in this case. Here, we can clearly see the positive lobes (shown in green) as well as the negative lobes.....12

Figure 9 Two-dimensional filter – In this two-dimensional filter each row is comprised of an array with values of second derivative of a Gaussian with $\boldsymbol{\sigma = 4}$, the size of this filter is $(\boldsymbol{6\,\sigma + 1}) \times (\boldsymbol{6\,\sigma + 1})$....13

Figure 10 The modified two-dimensional filter. To obtain this filter, the filter in the previous part is multiplied by a normalized 1D Gaussian in the y-direction....14

Figure 11 Rotated two-dimensional filters....15

Figure 12 The original and RGB Skyscraper image 16

Figure 13 The results obtained from applying each rotated filter to the gray scaled image 17

Figure 14 Four sets of zero-crossings, each zero-crossing pattern is obtained by applying a specific rotated filter. The blue lines show the direction of the second derivative. The obtained edges are in line with the direction of the oriented second derivative filter. 18

Figure 15 Edge detection by using multiple directional second derivatives 19

Figure 16 Studying the effect of standard deviation on the pattern of found edges....20

Figure 17 Testing the effect of variables (in this case standard deviation) on the resulted zero-crossing maps (detected edges)....21

Figure 18 Flow chart of general algorithm for Laplacian of Gaussian operator [4]....23

Figure 19 The Laplacian filter....24

Figure 20 Edge detection using a Laplacian filter 25

Figure 21 Important lines in the skyscraper image $\boldsymbol{\Delta r = 1, \Delta\theta = 1, Cth = 10}$ 28

Figure 22 Important lines in the railroad image $\boldsymbol{\Delta r = 1, \Delta\theta = 1, Cth = 10}$....28

Figure 23 Important lines in the skyscraper image $\boldsymbol{\Delta r = 1, \Delta\theta = 3, Cth = 10}$ 28
Figure 24 Important lines in the railroad image $\boldsymbol{\Delta r = 1, \Delta\theta = 3, Cth = 10}$.....28
Figure 25 Important lines in the skyscraper image $\boldsymbol{\Delta r = 1, \Delta\theta = 1, Cth = 5}$.....29
Figure 26 Important lines in the railroad image $\boldsymbol{\Delta r = 1, \Delta\theta = 1, Cth = 5}$ 29
Figure 27 Important lines in the skyscraper image $\boldsymbol{\Delta r = 3, \Delta\theta = 1, Cth = 10}$ 29
Figure 28 Important lines in the railroad image $\boldsymbol{\Delta r = 3, \Delta\theta = 1, Cth = 10}$.....29
Figure 29 A sample line fitting by using the total least square strategy.....30
Figure 30 Important lines in the skyscraper image $\boldsymbol{\Delta r = 1, \Delta\theta = 1, Cth = 10}$ 31
Figure 31 Important lines in the railroad image $\boldsymbol{\Delta r = 1, \Delta\theta = 1, Cth = 10}$.....31

QUESTION NUMBER ONE - MARR-HILDRETH

PART A

Consider a scalar function $f(x, y)$ of two variables x and y, where x and y are variables that are associated with two *orthogonal* directions. The Laplacian of f is given by $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$, where we assume that f is at least twice differentiable. Show that the Laplacian is rotation invariant, i.e., the value of the expression does not depend on the actual choice of the orthogonal directions.

[*Hint: Consider an arbitrary orientation θ, with r a variable representing position on a line along that orientation. Then compute Δf using calculus and the chain rule, but using this line as one of the chosen coordinate directions.* To do this properly you need to introduce a change of coordinates.]

ANSWER – PART A

The ultimate goal is to show that Laplacian operator is rotation invariant:

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \frac{\partial^2 f}{\partial u^2} + \frac{\partial^2 f}{\partial v^2}$$

$$where: u = x\cos\theta + y\sin\theta \;\; \boldsymbol{and} \;\; v = -x\sin\theta + y\cos\theta$$

To prove the aforementioned equation, let us start with the left side of the equation $(\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2})$ and try to drive the left side $(\frac{\partial^2 f}{\partial u^2} + \frac{\partial^2 f}{\partial v^2})$.

Let us start with $\frac{\partial^2 f}{\partial y^2}$ and use the chain rule:

$$\frac{\partial^2 f}{\partial y^2} = \frac{\delta}{\delta y}\left(\frac{\delta f}{\delta y}\right) = \frac{\delta}{\delta y}\left(\frac{\delta f}{\delta u}\cdot\frac{\delta u}{\delta y} + \frac{\delta f}{\delta v}\cdot\frac{\delta v}{\delta y}\right) = \frac{\delta}{\delta y}\left(\frac{\delta f}{\delta u}\sin\theta + \frac{\delta f}{\delta v}\cos\theta\right)$$

$$\frac{\partial^2 f}{\partial y^2} = \frac{\delta}{\delta y}\frac{\delta f}{\delta u}\sin\theta + \frac{\delta}{\delta y}\frac{\delta f}{\delta v}\cos\theta$$

$$\frac{\delta}{\delta y}\frac{\delta f}{\delta u} = \frac{\delta}{\delta u}\frac{\delta f}{\delta y} = \frac{\delta}{\delta u}\left(\frac{\delta f}{\delta u}\cdot\frac{\delta u}{\delta y} + \frac{\delta f}{\delta v}\cdot\frac{\delta v}{\delta y}\right) = \frac{\partial^2 f}{\partial u^2}\sin\theta + \frac{\delta}{\delta u}\frac{\delta f}{\delta v}\cos\theta$$

$$\frac{\delta}{\delta y}\frac{\delta f}{\delta v} = \frac{\delta}{\delta v}\frac{\delta f}{\delta y} = \frac{\delta}{\delta v}\left(\frac{\delta f}{\delta u}\cdot\frac{\delta u}{\delta y} + \frac{\delta f}{\delta v}\cdot\frac{\delta v}{\delta y}\right) = \frac{\delta}{\delta v}\frac{\delta f}{\delta u}\sin\theta + \frac{\partial^2 f}{\partial v^2}\cos\theta$$

$$\frac{\partial^2 f}{\partial y^2} = \left(\frac{\partial^2 f}{\partial u^2}\sin\theta + \frac{\delta}{\delta u}\frac{\delta f}{\delta v}\cos\theta\right)\sin\theta + \left(\frac{\delta}{\delta v}\frac{\delta f}{\delta u}\sin\theta + \frac{\partial^2 f}{\partial v^2}\cos\theta\right)\cos\theta$$

Let us start with $\frac{\partial^2 f}{\partial x^2}$:

$$\frac{\partial^2 f}{\partial x^2} = \frac{\delta}{\delta x}\left(\frac{\delta f}{\delta x}\right) = \frac{\delta}{\delta x}\left(\frac{\delta f}{\delta u}.\frac{\delta u}{\delta x} + \frac{\delta f}{\delta v}.\frac{\delta v}{\delta x}\right) = \frac{\delta}{\delta x}\left(\frac{\delta f}{\delta u}\cos\theta + \frac{\delta f}{\delta v}(-\sin\theta)\right)$$

$$\frac{\partial^2 f}{\partial x^2} = \frac{\delta}{\delta x}\frac{\delta f}{\delta u}\cos\theta - \frac{\delta}{\delta x}\frac{\delta f}{\delta v}\sin\theta$$

$$\frac{\delta}{\delta x}\frac{\delta f}{\delta u} = \frac{\delta}{\delta u}\frac{\delta f}{\delta x} = \frac{\delta}{\delta u}\left(\frac{\delta f}{\delta u}.\frac{\delta u}{\delta x} + \frac{\delta f}{\delta v}.\frac{\delta v}{\delta x}\right) = \frac{\partial^2 f}{\partial u^2}\cos\theta - \frac{\delta}{\delta u}\frac{\delta f}{\delta v}\sin\theta$$

$$\frac{\delta}{\delta x}\frac{\delta f}{\delta v} = \frac{\delta}{\delta v}\frac{\delta f}{\delta x} = \frac{\delta}{\delta v}\left(\frac{\delta f}{\delta u}.\frac{\delta u}{\delta x} + \frac{\delta f}{\delta v}.\frac{\delta v}{\delta x}\right) = \frac{\delta}{\delta v}\frac{\delta f}{\delta u}\cos\theta - \frac{\partial^2 f}{\partial v^2}\sin\theta$$

$$\frac{\partial^2 f}{\partial x^2} = \left(\frac{\partial^2 f}{\partial u^2}\cos\theta - \frac{\delta}{\delta u}\frac{\delta f}{\delta v}\sin\theta\right)\cos\theta - \left(\frac{\delta}{\delta v}\frac{\delta f}{\delta u}\cos\theta - \frac{\partial^2 f}{\partial v^2}\sin\theta\right)\sin\theta$$

Now, we can conclude about "$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$":

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \left(\frac{\partial^2 f}{\partial u^2}\cos\theta - \frac{\delta}{\delta u}\frac{\delta f}{\delta v}\sin\theta\right)\cos\theta - \left(\frac{\delta}{\delta v}\frac{\delta f}{\delta u}\cos\theta - \frac{\partial^2 f}{\partial v^2}\sin\theta\right)\sin\theta + \left(\frac{\partial^2 f}{\partial u^2}\sin\theta + \frac{\delta}{\delta u}\frac{\delta f}{\delta v}\cos\theta\right)\sin\theta + \left(\frac{\delta}{\delta v}\frac{\delta f}{\delta u}\sin\theta + \frac{\partial^2 f}{\partial v^2}\cos\theta\right)\cos\theta$$

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \frac{\partial^2 f}{\partial u^2}(\cos^2\theta + \sin^2\theta) + \frac{\delta}{\delta v}\frac{\delta f}{\delta u}(-\sin\theta\cos\theta - \cos\theta\sin\theta + \cos\theta\sin\theta + \sin\theta\cos\theta) + \frac{\partial^2 f}{\partial v^2}(\sin^2\theta + \cos^2\theta)$$

For any possible value of θ → $\cos^2\theta + \sin^2\theta = 1$

And also: $-\sin\theta\cos\theta - \cos\theta\sin\theta + \cos\theta\sin\theta + \sin\theta\cos\theta = 0$

So, we can rewrite the equation as below:

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \frac{\partial^2 f}{\partial u^2} + \frac{\partial^2 f}{\partial v^2} \rightarrow Laplacian\ is\ rotation\ invarient.$$

PART B

Edge detection in 2D is in general computationally expensive because a priori we do not know the local orientation of an edge. Hence, one has to look for zero-crossings of a second directional derivative, taken in multiple local directions. This is in fact how the mammalian visual system implements a form of edge detection in particular layers of the visual cortex. In their paper on edge detection Marr and Hildreth get around this complexity by associating edge locations with zero-crossings of the non-directional (as in rotation invariant) operator, the Laplacian. They claim that this operation picks out the direction which has highest slope of the zero-crossing (if one were to take directional derivatives that is). However, they make an important assumption about the local intensity profile. *What is that assumption? Explain briefly how, if that assumption holds, their argument works.*

ANSWER – PART B

Let us start with the first approach: For each point in a specific local area, we simply calculate the second derivatives in multiple different directions, if there is an edge segment nearby, all the directional second derivatives will have zero-crossings expect the derivative which is parallel to the orientation of the edge, and these zero-crossing all lie on a line which is the edge segment. In this approach the amplitude of the obtained zero-crossings, I mean the difference between the positive and negative lobes, is important; actually, the second derivative which has the best direction shows the most powerful zero-crossing, I mean in this best case the amount of variation between the positive and negative points is more than the amount of zero-crossings in other directions. And this best direction is actually perpendicular to the edge segment which exists in the local area.

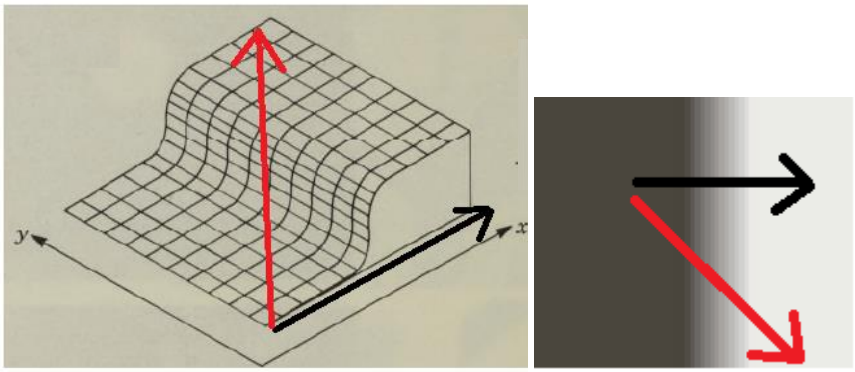


Figure 1 Suppose this is your intensity profile near the edge, the red line is the direction of one of the second derivatives and the black line is the direction of another second derivative, in this case the black direction is the one known as the best direction, since when looking at the profile of second derivatives, this directional second derivative has the strongest zero-crossing which means in this case the slope of zero-crossing is the largest among other directional derivatives.

Figure 2 This is the profile of the second directional derivative in the red direction. Here, the slope of zero-crossing is not that sharp.

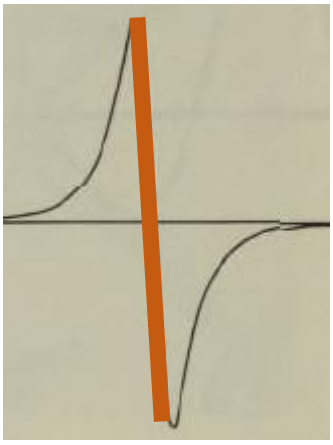


Figure 3 This is the profile of the second directional derivative in the black direction. Here, the slope of zero-crossing is sharp.

Although, here we have an assumption which says that on the **edge segment or any direction parallel to the edge the intensity should vary linearly which means the intensity in constant or there is a ramp-like change in intensity**. If we have a sharp nonlinear change in the intensity on the edge segment or on any parallel line near the edge segment, we might not be that successful in localizing the edge segment, the reason is that if there is a nonlinear change in intensity, when calculating the second derivatives, we might get a powerful zero-crossing in a direction that has raised from the existence of that specific nonlinear change; actually, this nonlinear change can lead to finding a false best direction; hence, the derived edge segment is not well positioned. Therefore, the detectability issue remains if we want to look at the strongest response. However, the abovementioned **constraint is valid most of the time when we look at any real-world scene**, **locally parallel to an edge, the intensity looks constant.**

Marr and Hildreth need the condition of **linear variation** to prove their theorem. They claim that under the condition of linear variability the Laplacian is a simple way to find edge segments without calculating all the directional derivatives. First, they assume some smoothing, once the image is smooth, they are talking about smooth results and they claim that using the Laplacian – which is an orientation-independent second-order differential operator - can be a used for finding edge segments. Actually, the method operated by convolving the image with the Laplacian of the Gaussian function ($\nabla^2 G * I$) and then zero-crossings are detected in the filtered result to obtain edge segments. In their strategy, they **assume that intensity variation in $\boldsymbol{G * I}$ is linear along but not necessarily near to a line of zero-crossing**. Under this certain condition, the zero-crossings of $\nabla^2 G * I$ can illustrate the so-called edge segments. If intensity varies

along a segment in a very nonlinear way, the Laplacian, and hence the operator $\nabla^2 G$ will see the zero-crossing displaced to one side [1,2].

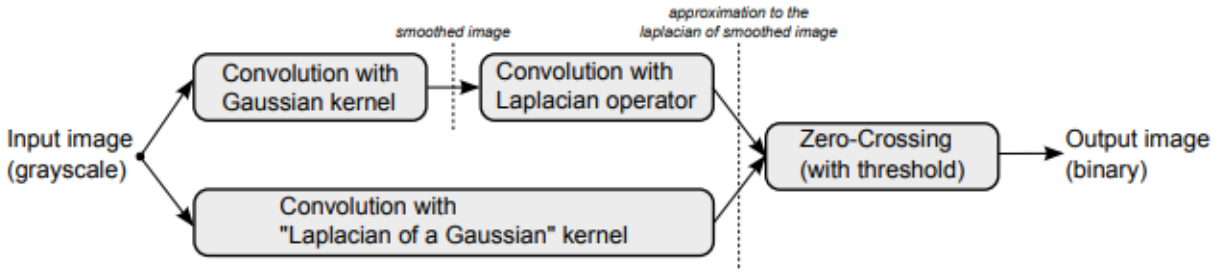


Figure 4 Block diagram of the Marr-Hildreth algorithm [2]

Let us consider a case when **liner variation constraint no longer exists**; here, there is a **corner** like below, you can see that the intensity change is different in the two sides of the corner:

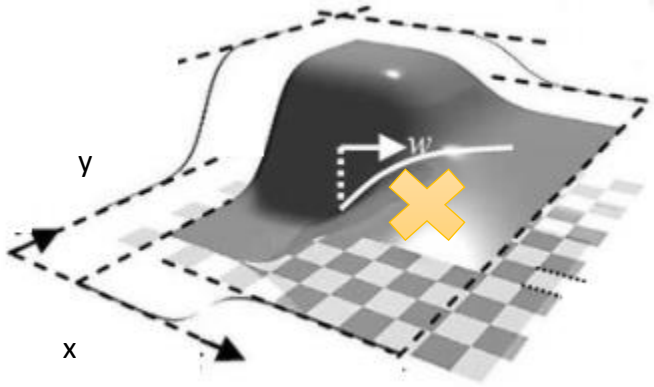


Figure 5 Example of a situation where linear variation constraint is not met (a corner)

We will probably face a problem in this case, if we start calculating the second derivatives in the x direction and step by step move toward the y direction, in some point, we get to the corner and then there is a huge intensity change on the other side of this corner, here we have a strong zero-crossing which is more powerful (have a sharper slope) than the best orientation in the x direction that we wanted to find, so we will miss one of the edge segments (it is illustrated by using a yellow cross on it). Since the direction of the edge changes instantly, corner pixels look in the wrong directions for its neighbors. Hence, the Marr-Hildreth **edge detector fails to detect many prominent edges.**

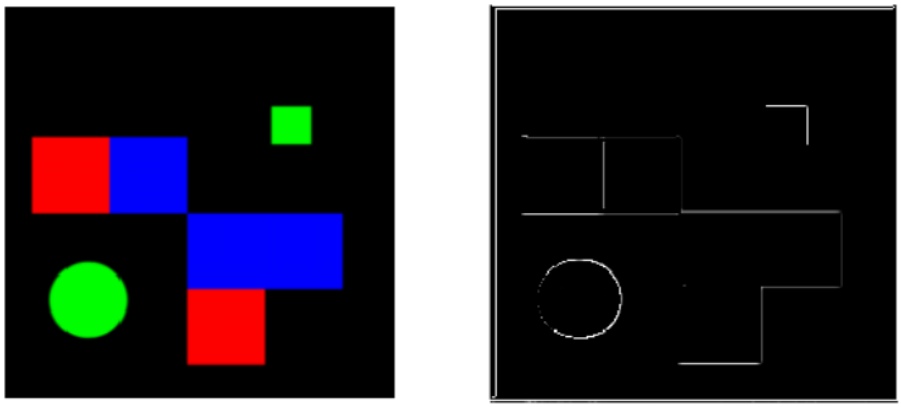
PART C

Given your answer in part b) when would you expect the Marr-Hildreth edge detection strategy to give good results and when you would expect it to fail? You can draw on your intuition from a natural image of a complex scene, which you could present and use to defend your argument. *You don't have to implement the strategy, you just have to provide your reasoning.*

ANSWER – PART C

(For complete explanation, please also consider my answer for part b.)

When the intensity variation in $G * I$ is linear, which means the edge segment is linear in the real world, it should work well. This condition is approximately true in smoothed practical images. However, when there is a nonlinear change of intensity along an edge, the Laplacian method is not going to work appropriately. If we have a vertical step edge modulated vertically by a nonlinear function; the intensity variation along the edge is not linear.



Original Image Marr-Hildreth Edge Detector

Figure 6 Example of a case when Marr-Hildreth edge detector might collapse [3]

QUESTION NUMBER TWO - EDGE DETECTION USING ORIENTED OPERATORS

PART A

In class we've seen 1D Gaussian's and their derivatives. Let σ be an integer variable that is associated with the standard deviation of a 1D Gaussian. Create a 2D array with dimensions

$(2\,\sigma + 1)$ x $(2\,\sigma + 1)$. The centre of the array represents the (x, y) position (0,0). Now populate the entries at each row in the array with the value of the second derivatives of a 1D Gaussian in the x direction, with standard deviation σ. The 1D Gaussian should have its peak value in the middle of the row, and the values in each row should be identical. The original Gaussian should also be normalized to have area equal to 1 underneath it, prior to the computation of the second derivative. *Show an image of this filter and show the colormap using the* `colorbar` *command. Also visualize the filter values as a 2D height function, using Matlab's surface plotting functions* (e.g., `surf`, `surfc`, `mesh`, etc.). As an example, if you had a 2D Gaussian and you computed its partial derivatives in the x and y directions, a visualization of the results would look something like the following. (Ofcourse, the filter you are designing in this question is not the same thing.)

ANSWER – PART A

In the first step, a normalized one-dimensional Gaussian is implemented using the `fspecial` comment in MATLAB. The standard deviation of this Gaussian is equal to $\sigma = 4$. In the next step, the second derivative of the Gaussian is derived by convolving the constructed one-dimensional Gaussian with the central derivative filter twice (central derivative filter is: $[-\frac{1}{2}, 0, \frac{1}{2}]$). It should be noted that when implementing the Gaussian filter, filter size of $(2\,\sigma + 1) + 4$ was considered, since I wanted to compensate for the effect of zero-padding happening in the convolution steps. You can observe the Gaussian and its second derivative in Figure 7.

It is evident that if the study's goal is to design a filter that could be used for edge detection in the image. A straightforward strategy for edge detection is to apply the second derivative of the gaussian filter to the image and to find the zero-crossing. However, using this filter size is not elegant, the reason behind this claim is that in Figure 7, the second derivative profile is just like an inverse Gaussian, **there is no positive lobes seen in the profile**, which means that if we apply this filter to an image, we are not going to observe any meaningful zero-crossing and **subsequently we are not going to detect any edge**. One way to solve the problem is to extent the so-called field of view, meaning that we can just improve the $(2\,\sigma + 1)$ limitation for the filter size, and change it to $(6\,\sigma + 1)$. In this case, when we look into the profile of the second derivative of Gaussian, we can clearly observe the negative and the positive lobes in Figure 8.

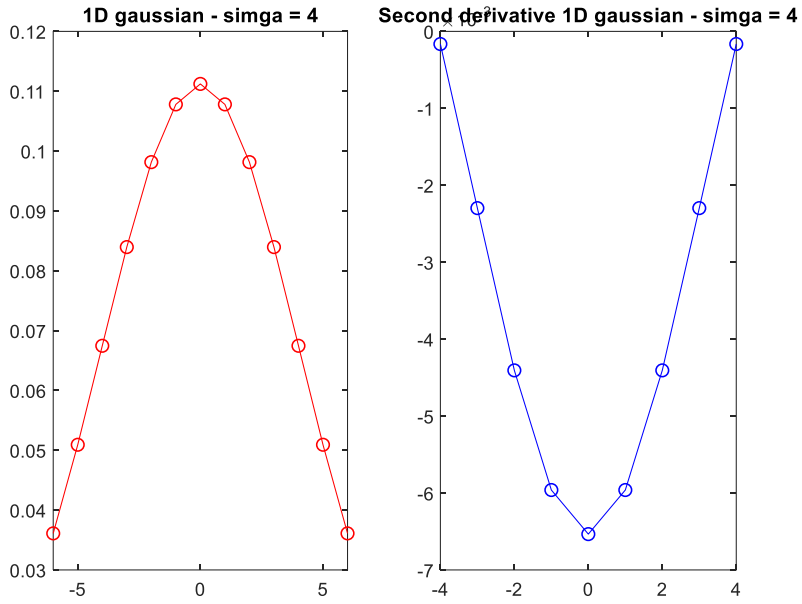


Figure 7 Gaussian filter is shown in red and its second derivative is illustrated in blue. Size of array for the second derivative of Gaussian is equal to $(2\,\sigma + 1)$ in this case.

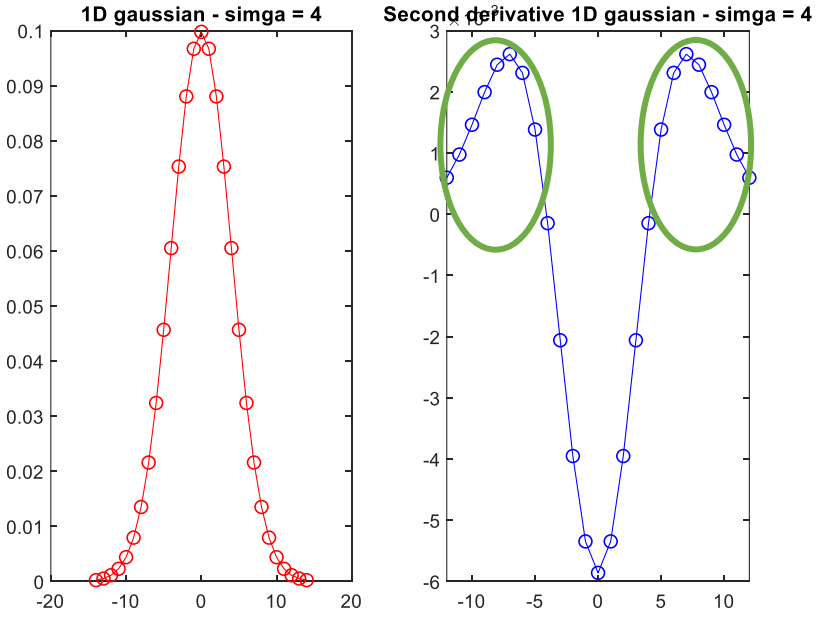


Figure 8 Gaussian filter is shown in red and its second derivative is illustrated in blue. Size of array for the second derivative of Gaussian is equal to $(6\,\sigma + 1)$ in this case. Here, we can clearly see the positive lobes (shown in green) as well as the negative lobes.

Now that we have implemented the primary pattern in one dimension, we may concatenate several one-dimensional filters of this model to form a two-dimensional filter. We obtained this

array by using the "`repmat`" comment in MATLAB ; when doing so, we will obtain a filter shown in Figure 9.

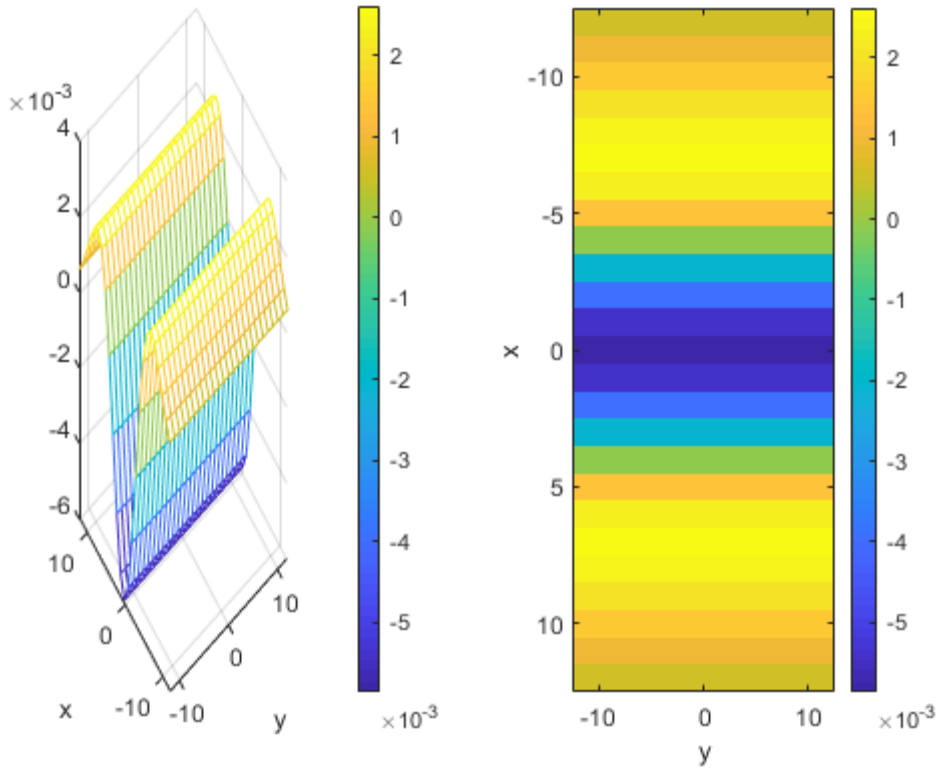


Figure 9 Two-dimensional filter – In this two-dimensional filter each row is comprised of an array with values of second derivative of a Gaussian with $\sigma = 4$, the size of this filter is $(6\sigma + 1) \times (6\sigma + 1)$

PART B

Now modify the filter above, by multiplying its entries by a normalized 1D Gaussian in the y direction, with standard deviation σ_y, with $\sigma_y > \sigma$. This should have the effect of tapering the filter values in the y direction. *Show an image of this modified filter and show the colormap using the* `colorbar` *command. If you like you can also visualize it using Matlab's surface plotting functions* (`surf, surfc, mesh`, etc.).

ANSWER – PART B

In this step, the obtained two-dimensional filter in the previous section is multiplied by a normalized one-dimensional Gaussian in the y-direction. The standard deviation of this one-dimensional Gaussian is two times more than the standard deviation of this one-dimensional Gaussian used in the previous section. ($\sigma y = 2\sigma = 8$). Note that in this case the filter size is still preserved as $(6\sigma + 1) \times (6\sigma + 1)$. The tapering effect is obviously present in the results.

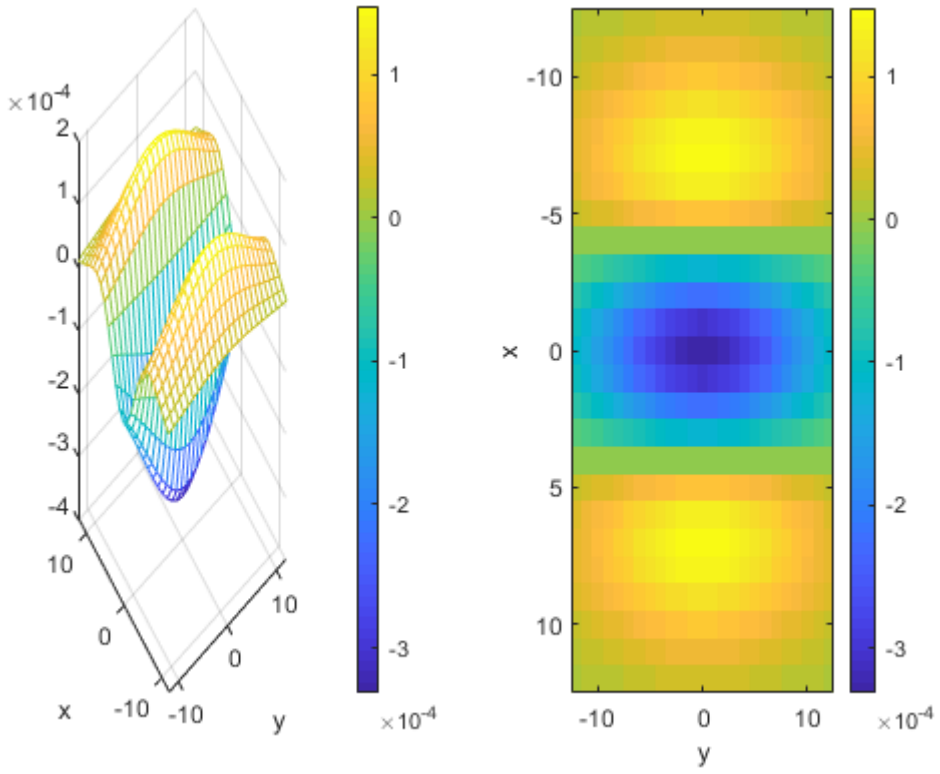


Figure 10 The modified two-dimensional filter. To obtain this filter, the filter in the previous part is multiplied by a normalized 1D Gaussian in the y-direction.

PART C

You will now write a routine to create a rotated version of the 2D filter in part b). The goal is to rotate it by a small amount θ. An effective way of doing this is just to rotate the coordinates. In other words, for the value at a coordinate (x', y'), in the destination 2D filter, where these coordinates represent a rotation of (x, y) by θ about the origin, we simply copy the value from the source 2D filter, at location (x, y). *Show images of this modified filter for different* $\theta = 0, 45, 90$ and 135 $degrees$*, including the colormap using the* `colorbar` *command. Also visualize the rotated filters as 2D height functions, as you did for part b).*

ANSWER – PART C

Let us start with some basic concept of rotation and discuss it mathematically. Suppose we have a point located at (x, y) and suppose that we want to rotate this point by θ degree. To do so, let us describe the new location as (x', y'), in this case, we can obtain the values of x' and y' by using the equations described below:

$$x' = xcos\theta - ysin\theta$$

$$y' = xsin\theta + ycos\theta$$

In this case, to obtain the rotated filter we followed this routine:

- For each pixel at position (x, y), we found the corresponding position in the rotated version (x', y').
- If this (x', y') falls **within** the original filter array ($xmin \leq x' \leq x\max \ \ and \ ymin \leq y' \leq y\ max$), we simply dedicate the value in (x, y) position to its corresponding rotated pixel (x', y').
- If this (x', y') falls **beyond** the original filter array size, we have to use **interpolation** to decide for the NAN pixels.

By applying this technique, we got the following results, as illustrated in Figure 11.

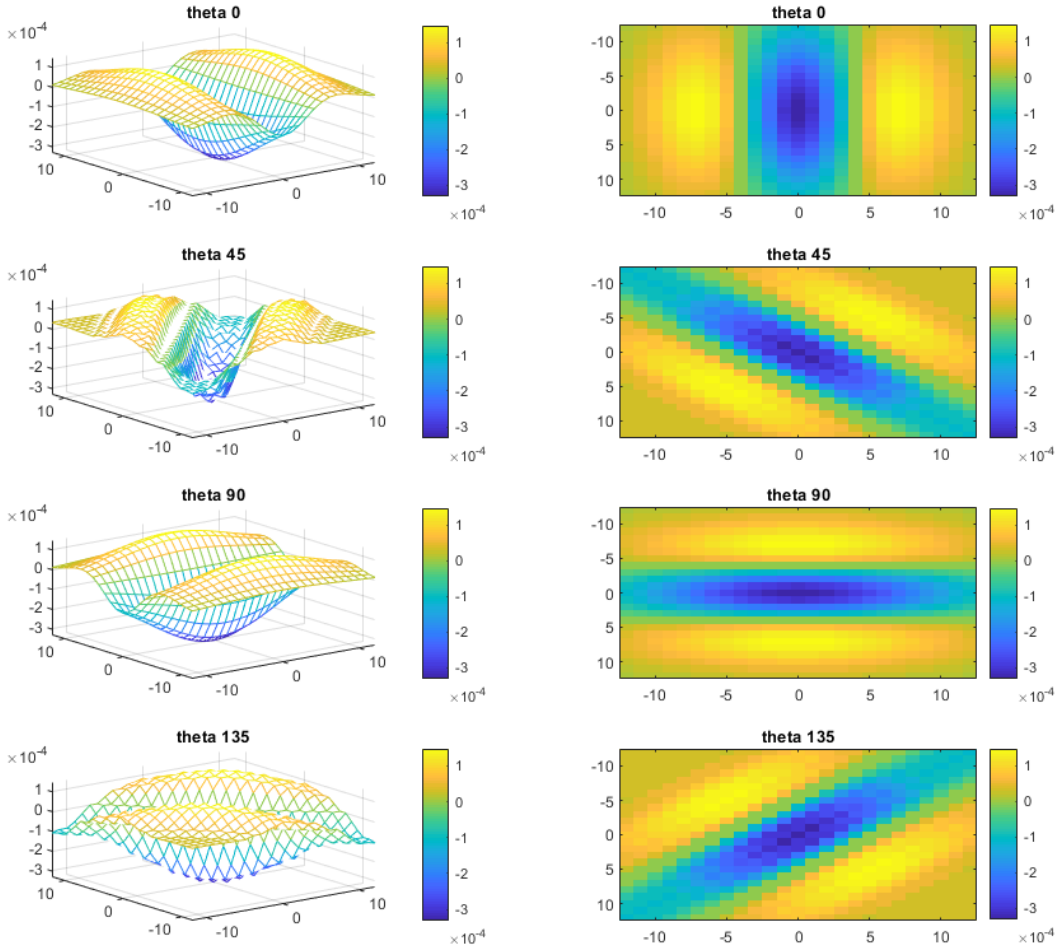


Figure 11 Rotated two-dimensional filters

PART D

We have provided an image of skyscrapers in Manhattan, viewed under perspective projection. Filter this image with your filter from c), but using the 4 rotated copies of it (0,45,90 and 135 degrees). For each copy, find the pixels at which there is a zero crossing in either the x or y or diagonal directions. Adjust your choice of σ so that you get acceptable results. Create a binary image that indicates where these zero crossing points are. Note: you will actually have 4 sets of

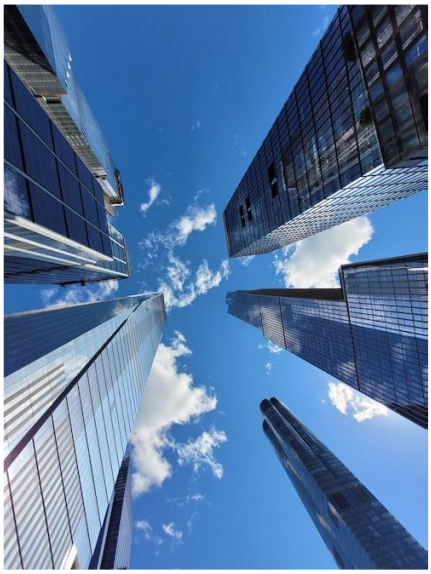
zero-crossings, one for each orientation of the filter so you can think of a clever way of presenting these results together. One idea could be to show each set of zero-crossings in a different colour and then use the *Matlab functions such as* `imfuse` *to create a composite image. Alternatively, you can use different colours for each filter orientation. You just have to explain your visualization.*

For this part you have some flexibility in how you choose to display your results. You might need to tune your parameters σ, σ_y. Do not use `for loops` to find the zero crossings. Instead, use vectorization. Hint: You want to compare neighboring pixels of the filtered image and see where the values have different sign, namely if their product is negative. To avoid the annoyances of image boundaries, use Matlab's `circshift` operation to compare an image to a shifted version of itself. Note that logical operations can also be vectorized.

ANSWER – PART D

In this section, we applied each of the implemented filters to the image of the skyscraper. In this regard, we first converted the colored image into a gray image using `rgb2gray` command.

Original picture



Black and white picture

Figure 12 The original and RGB Skyscraper image

And then, we chose a filter and applied it to the grayscale image (using convolution operator).

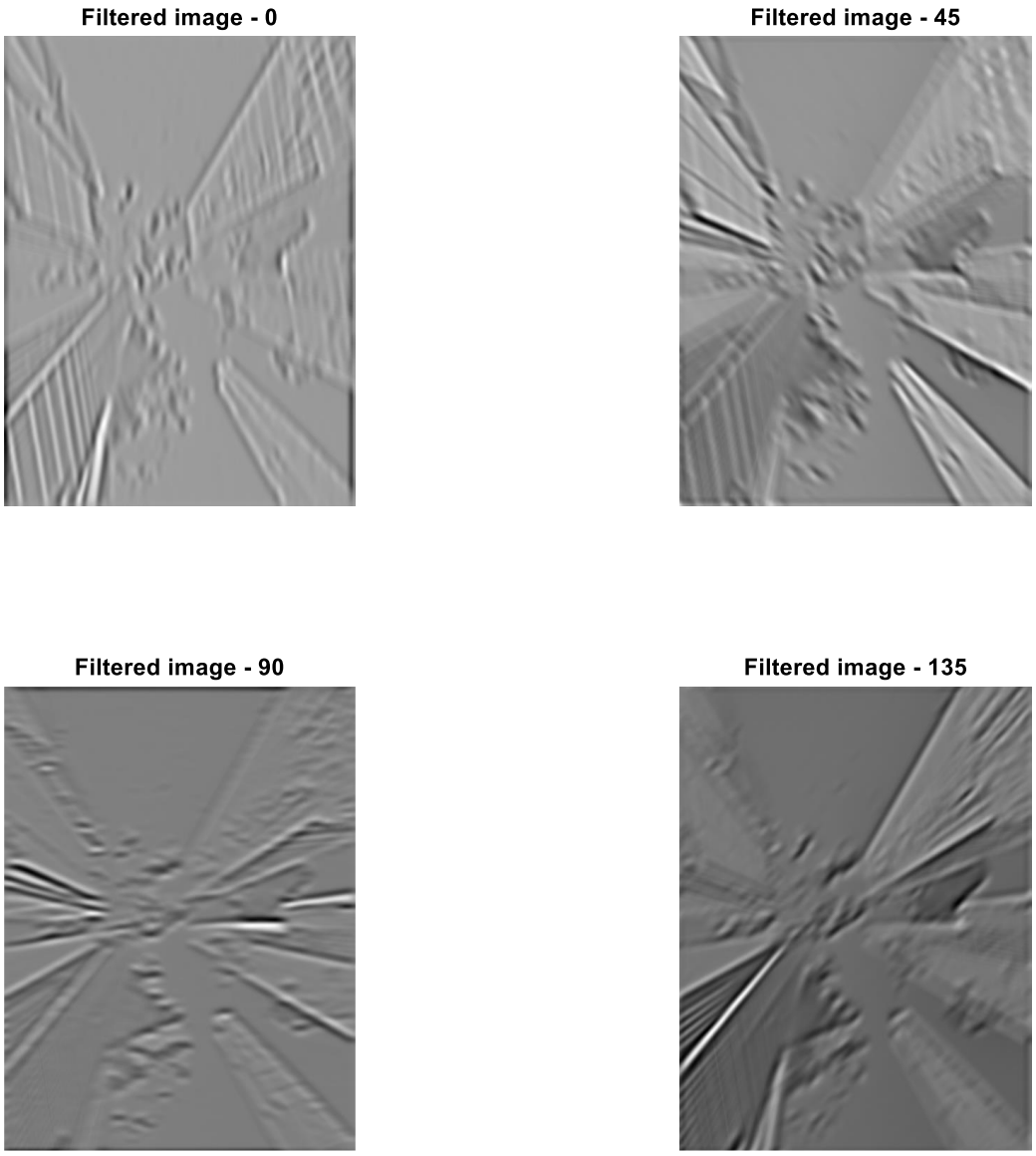


Figure 13 The results obtained from applying each rotated filter to the gray scaled image

In the next step, we tried to find zero-crossings in the x and y-direction. These zero-crossings are located where we have sharp intensity change in the image (these sharp changes also include edges in the image). But the question is: how can we find these zero-crossings? The answer is by using the shift function as bellow:

```
% If the intensity sign at a specific pixel is different from its neighbors in x or y direction mark the pixel as a location where zero-crossing is happening
temp(:,:,1) = sign(circshift(Filtered_img,-1,1)) ~= sign(Filtered_img);
temp(:,:,2) = sign(circshift(Filtered_img,-1,2)) ~= sign(Filtered_img);
zero-crossings = temp(:,:,1) + temp(:,:,2);
```

In Figure 14 you can see the pattern of the zero-crossings obtained**. Each filter has been successful in finding zero-crossing in a specific direction** (the same direction as the direction of the filter).

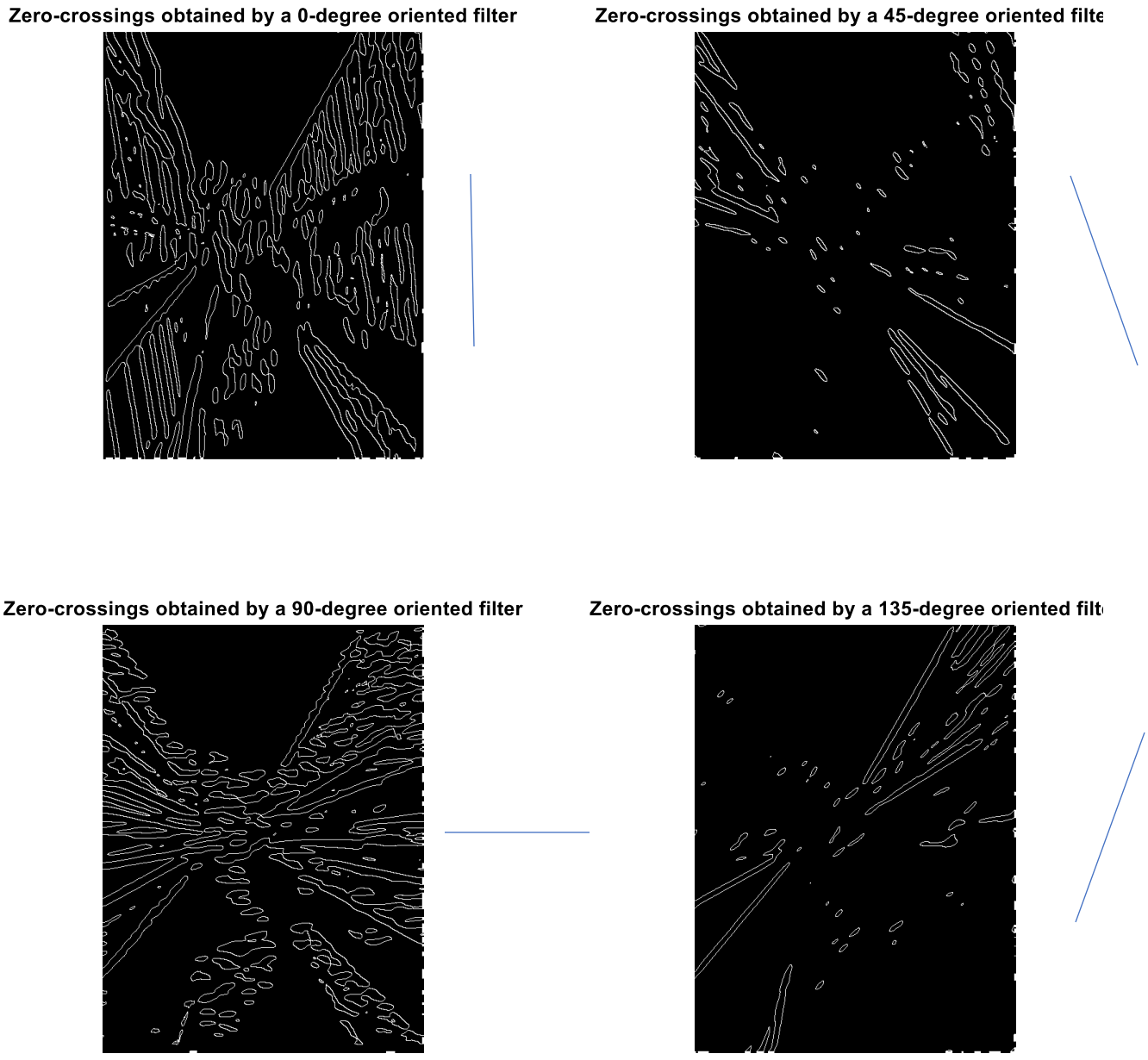


Figure 14 Four sets of zero-crossings, each zero-crossing pattern is obtained by applying a specific rotated filter. The blue lines show the direction of the second derivative. The obtained edges are in line with the direction of the oriented second derivative filter.

Here, we have calculated second derivatives in multiple directions and have found the corresponding zero-crossings in each direction. We should also consider that if the ultimate goal of the study is edge finding, we have to do multiple second-derivatives in different

orientations and find the corresponding zero-crossings in different directions; the more the **number of checked orientations, the more accurate the results would be.**

In the next step, we just **concatenated the results obtained from the previous step** to have a more precise and complete edge detection. As you can observe, we are now capable of finding edges in multiple locations. **We can obtain a more detailed edge pattern if we recruit filters in more directions.**

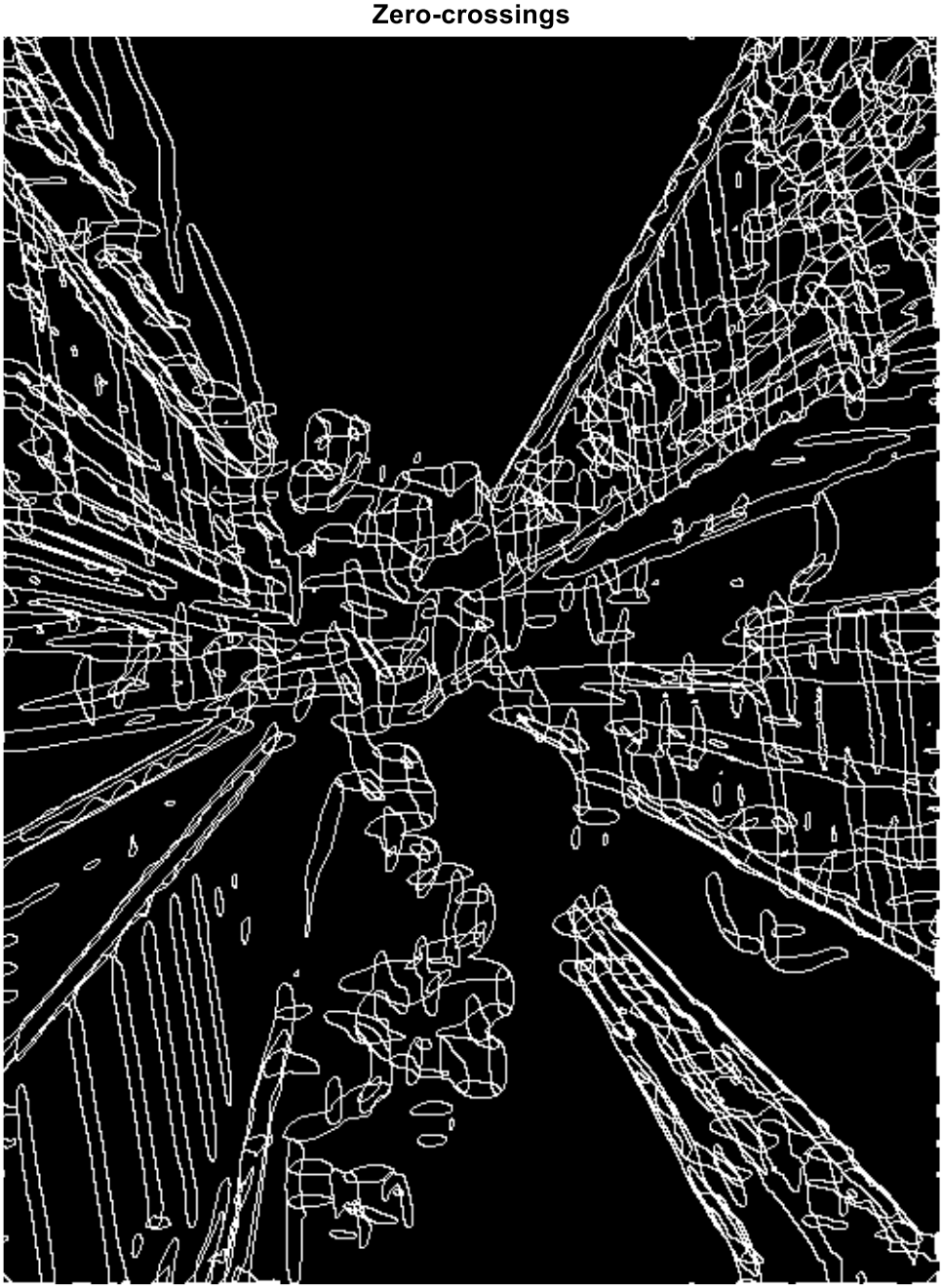


Figure 15 Edge detection by using multiple directional second derivatives

In the next step, I want to check the effect of changing **standard deviation** (σ). We should first keep in mind that:

- A serious practical problem with any edge detector is the matter of choosing the scale of smoothing (standard deviation).

- For many applications, it is desirable to be able to process an image at multiple scales.

- We determine which edges are most significant in terms of the range of scales over which they are observed to occur.

In the results, you can obviously see that the smaller the value of standard deviation would be, a smaller filter is obtained and we can get finer edge detection. You can see that if we increase standard deviation (σ), we will lose precision.

Decease the value of standard deviation (σ) to have finer details (more fine edges)

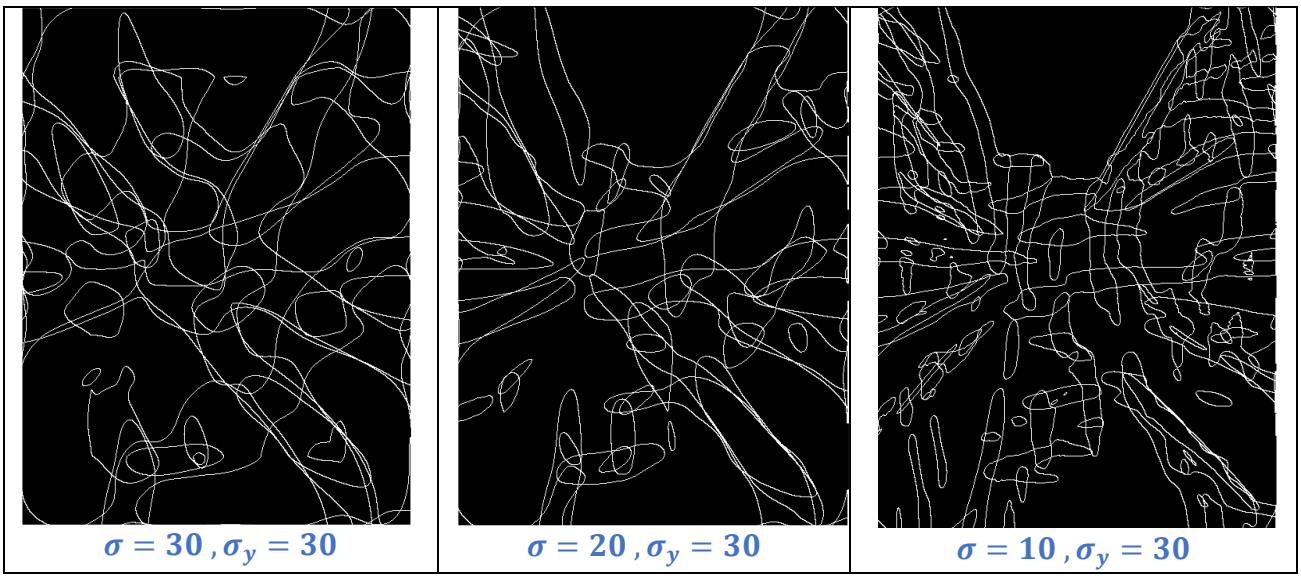


Figure 16 Studying the effect of standard deviation on the pattern of found edges

Generally, if you want to set a value for standard deviation, you are highly **dependent on the goal of your study**; one might need to find the large structures in the images and not be that much sensitive about having all the details, while the other researcher might set her/his goal to learn about the finest elements in the images. So, selecting an appropriate standard deviation might be somehow study-dependent. The only thing to keep in mind is that when we increase standard deviation while calculating the second derivate in a specific pixel, we are talking more information from neighboring pixels. Hence, the results are blurred. I have included some more examples in the following figure (Figure 17).

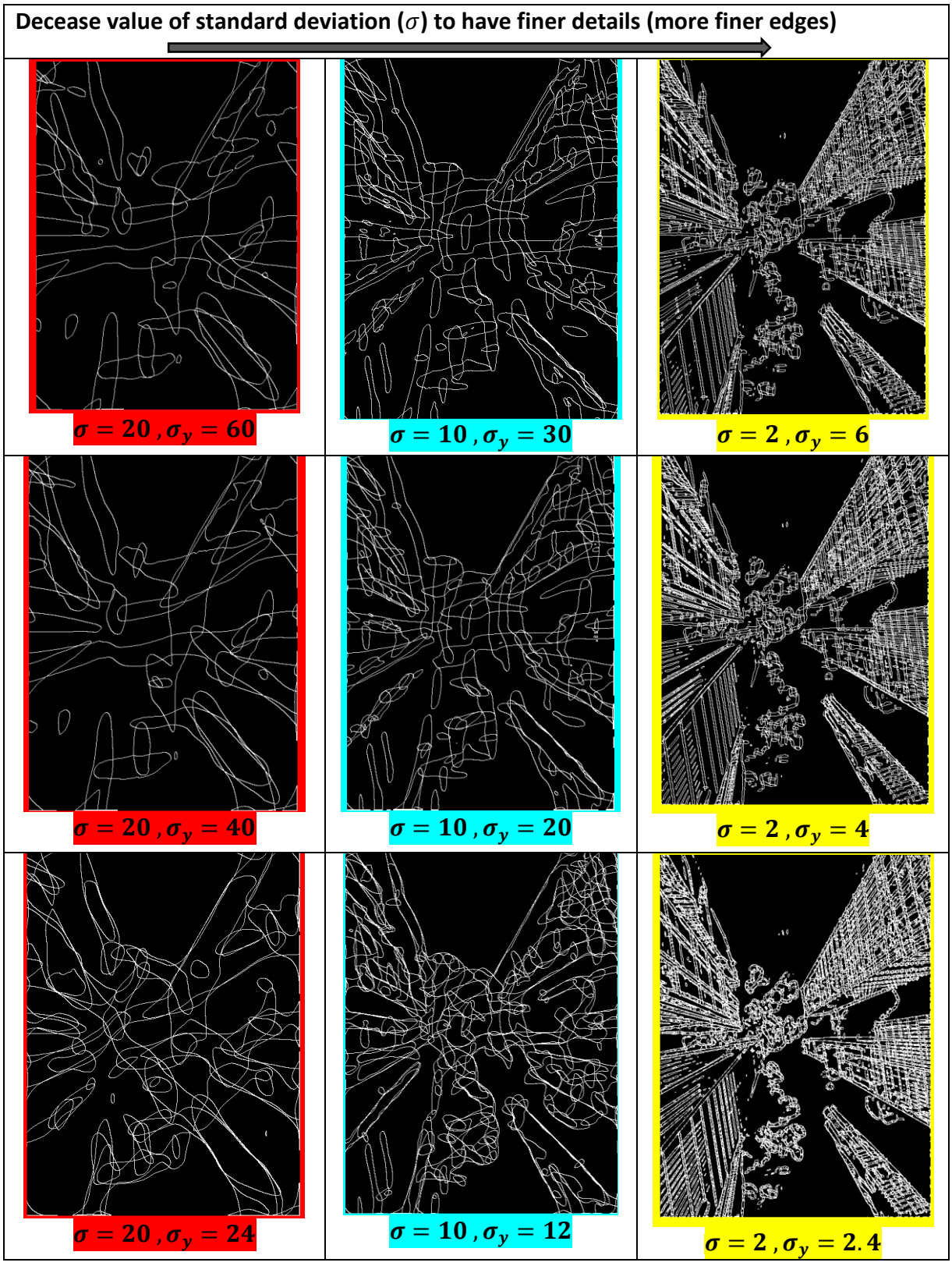


Figure 17 Testing the effect of variables (in this case standard deviation) on the resulted zero-crossing maps (detected edges)

PART E

Repeat (d) but this time use a Laplacian of a Gaussian filter with $\sigma = 6$ and width 40. To construct this filter you can directly use the Matlab function `fspecial`. *Briefly discuss how your results in e) compare with the results in d), using also your insights from question 1.*

ANSWER – PART E

In Figure 19 we have shown the figure used for Part E. It is a symmetric Laplacian filter, and it is orientation independent. The Laplacian $L(x, y)$ of an image with pixel intensity values $I(x, y)$ is given by:

$$L(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection zero-crossing edge detectors. The output from the zero-crossing detectors is usually a binary image with single-pixel thickness lines showing the positions of the zero-crossing points.

The important thing to emphasize is that when using **Laplacian is not dependent on direction.** So, when the goal is to find the edges, using **Laplacian can be more efficient in computation and time.** If we wanted to get the same results using a simple second derivative in multiple directions, we needed to go over the second derivative operation in many different directions, which is not efficient.

You can observe the algorithm used for this part of assignment in Figure 18 and also the Laplacian filter used in Figure 19.

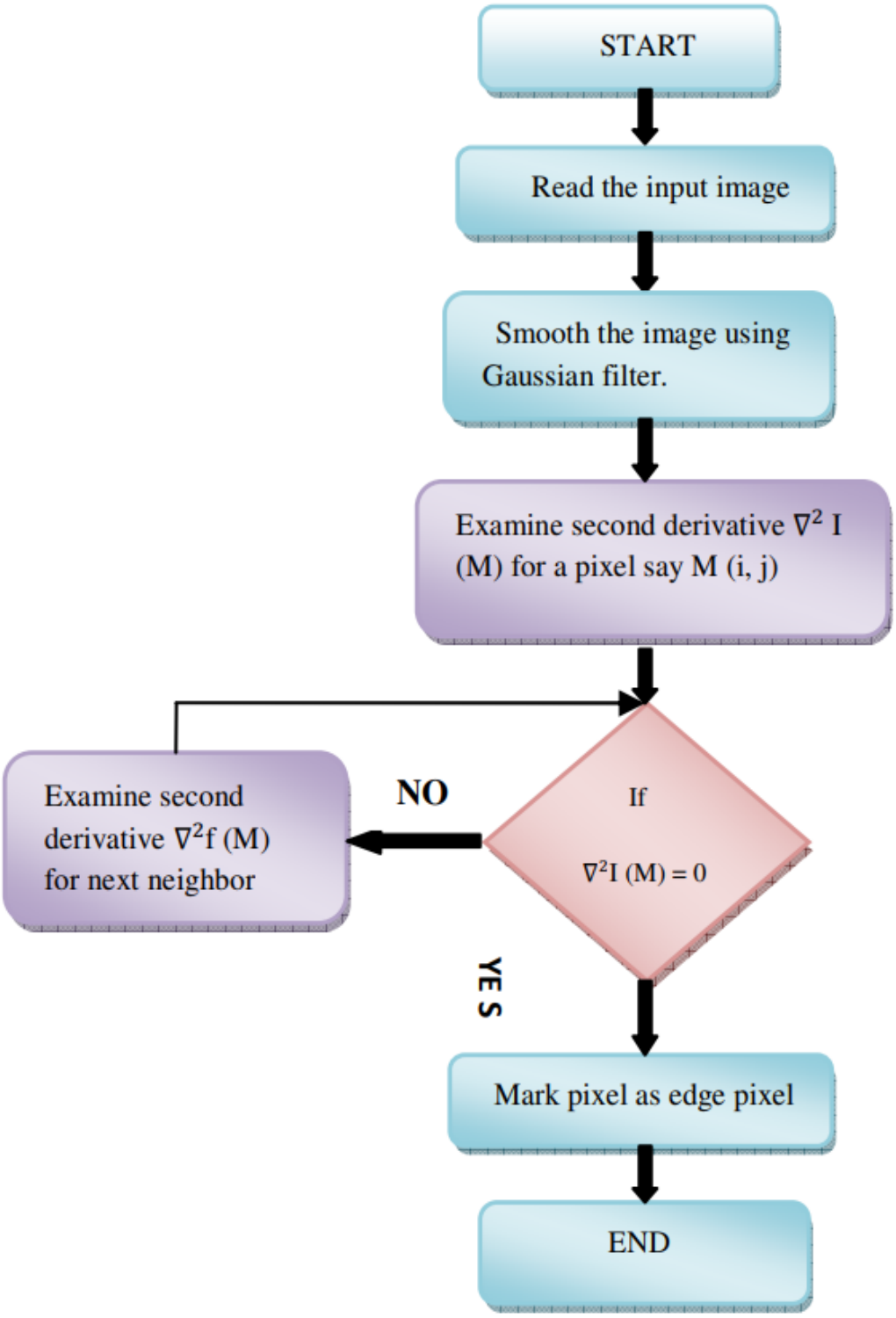


Figure 18 Flow chart of general algorithm for Laplacian of Gaussian operator [4]

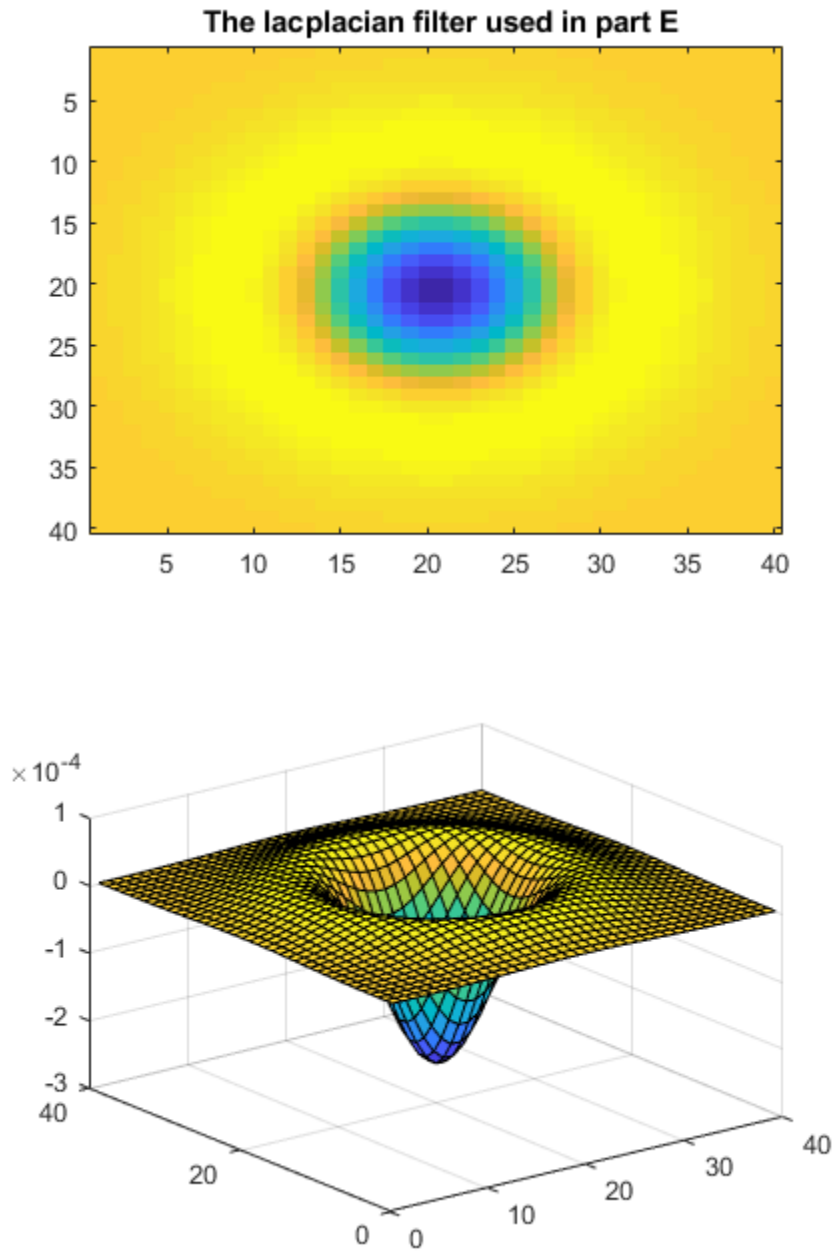


Figure 19 The Laplacian filter

This filter is generated by using the code below:

```
hsize = 40;
sigma = 6;
filter = fspecial('log',hsize,sigma);
```

The zero-crossing detector looks for places in the Laplacian of an image where the Laplacian value passes through zero. Such points often occur at edges in images where the intensity of the image changes rapidly. However, they also occur at places that are not as easy to associate with edges.

The filtered image - Part E

Zero-crossings obtained by using the Laplacian filter

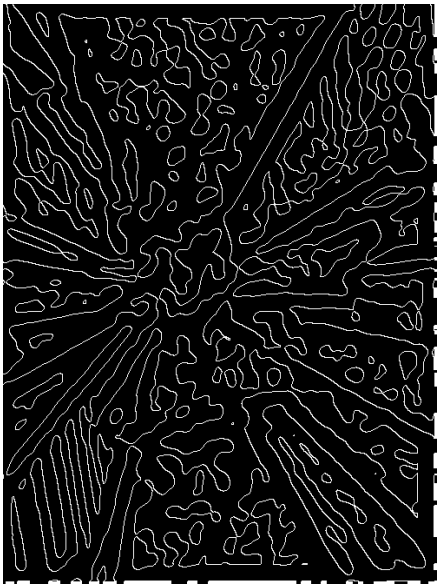


Figure 20 Edge detection using a Laplacian filter

QUESTION NUMBER THREE - LINE FINDING USING RANSAC

PART A

a) Your task is to implement the RANSAC method to find prominent lines in your input image. The pseudo-code for the RANSAC algorithm is in the slides and lecture notes provided in lecture 6, and it is presented in slightly simplified form here. In this simplified form, initially we are not using least squares to refine a fit, and we are not finding the 'best' model.

ANSWER – PART A

Here, I am going to describe the algorithm I used for implicating this part of the assignment.

As mentioned, we have the information about the edges detected by the Canny edge detector. This information includes the orientation of 8477 edge locations $(x, y, sin\theta, cos\theta)$.

1- We just used the `randperm` function changed the sequence of these 8477 points.

2- We found the corresponding r, for each of the edge locations :

```
r = xcos(t) + ysin(t)
```

3- We used the method described below to find the corresponding θ for each sample:

```
for i = 1:1:8477
    if ((sin>0 & cos>0) | (sin<0 & cos>0))
        theta = atan(sin/cos)
    end
    if ((sin>0 & cos<0))
        theta = atan(sin/cos) + pi
    end
    if ((sin<0 & cos<0))
        theta = atan(sin/cos) - pi
    end
end
```

So, to this end, for each edge location we have its representation in r and θ domain.

4- We can select the first edge location from the permuted list (so, the selection is entirely random.), So, we can have a r value and a θ value of this sample

5- Now, we need to check the remaining edges to see whether they are in line with the selected edge or not, but the question is: "How do you want to figure out if they are in line?"

To answer the abovementioned question, we simply define a threshold for difference of edge locations in terms of r (Δr) and another threshold for difference of edge locations in terms of θ ($\Delta\theta$). And now, we can find the edges that can meet this requirement:

```
((r_other-r_selected)<=dr)&(theta_other-theta_selected)<=dt))
```

6- We check the abovementioned constraint for all the available edge locations; we count the number of edge locations that have been able to meet the constraint and name this variable C.

7- If C is more significant than a threshold value (Cth), we calculate the average location using the data of the edges that we found are in line with the randomly selected sample. Then we save the information of this new line and discard all the lines that we have used to reach this critical line in the image. On the other side, if the C value is smaller than the threshold value, the line that we have chosen in the first place is not that powerful and cannot gain support from other lines in the image. Based on this information, I decided to remove the information of this selected sample from the data that I am going to use in the following steps. The reason is that I know that this sample is not going to work! It will probably not add any helpful information regarding the prominent lines in the image. So, to this point the permuted matrix of edge locations has some rows (if the sample can find matches and is considered as a powerful line) or a single row (if the sample was not considered a prominent line) removed, in both cases, the sample that was selected at the first place is removed.
8- This loop is going to be repeated multiple times. However, how is the number of repetitions chosen? Depending on defining the threshold values for our parameters, this number can vary. As I mentioned, if a sample gets lots of votes, the average line is calculated, and the data of all the in-line samples are removed; if the sample gets not that much support, again, what will happen is that we remove the data of that selected line. This means the size of our dataset is diminished in each iteration. I will continue the outer loop until I get to the point where all the samples have been used to make sure nothing is missed.

Note: Of course, this algorithm is not optimal, but it seems to work pretty well. I have used some different figures to check the validity of my work.

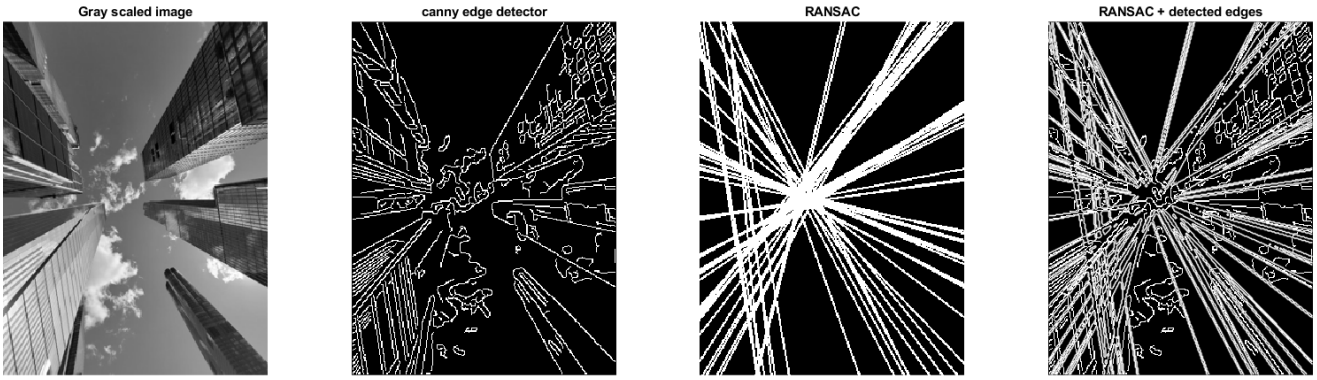


Figure 21 Important lines in the skyscraper image $\Delta r = 1, \Delta\theta = 1, Cth = 10$

Figure 22 Important lines in the railroad image $\Delta r = 1, \Delta\theta = 1, Cth = 10$

We can change the threshold value to obtain different results; in the following figures, I have changed the threshold on θ value, and we can see that here much more lines have emerged; this is because the threshold is more liberal in this case.

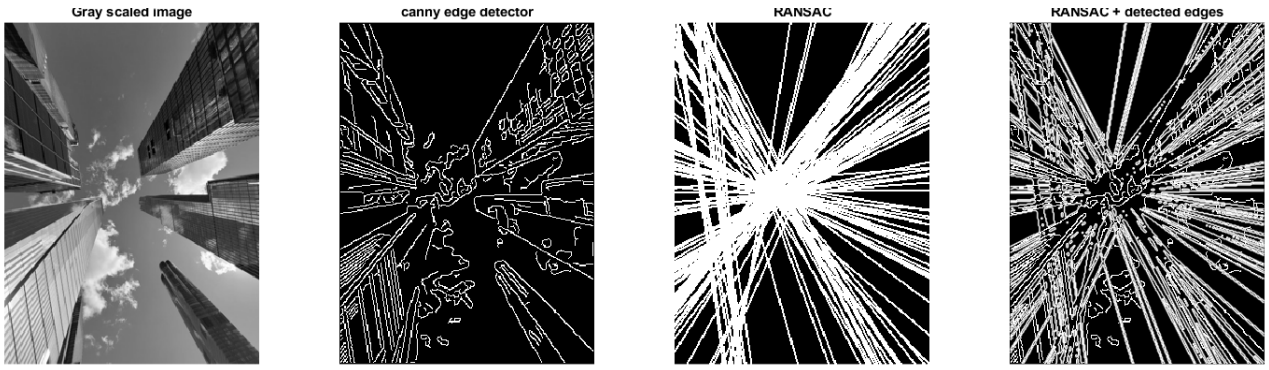


Figure 23 Important lines in the skyscraper image $\Delta r = 1, \Delta\theta = 3, Cth = 10$

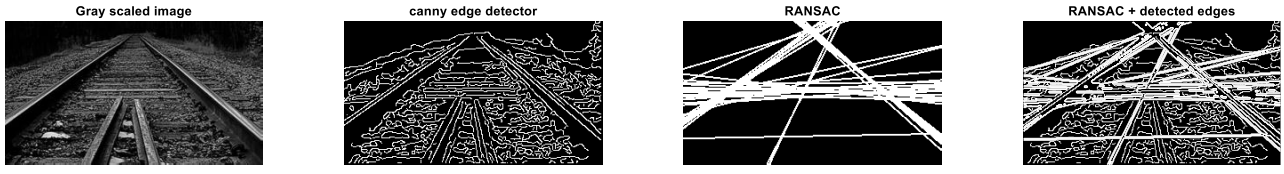


Figure 24 Important lines in the railroad image $\Delta r = 1, \Delta\theta = 3, Cth = 10$

Also, Cth is an essential factor; if we decrease it, we can be more liberal, so the number of important lines that are found will increase.

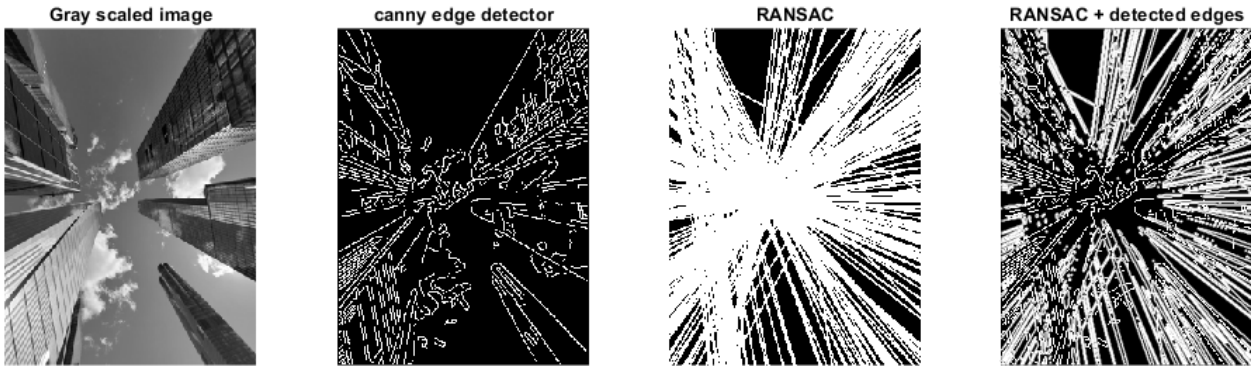


Figure 25 Important lines in the skyscraper image $\Delta r = 1, \Delta\theta = 1, Cth = 5$

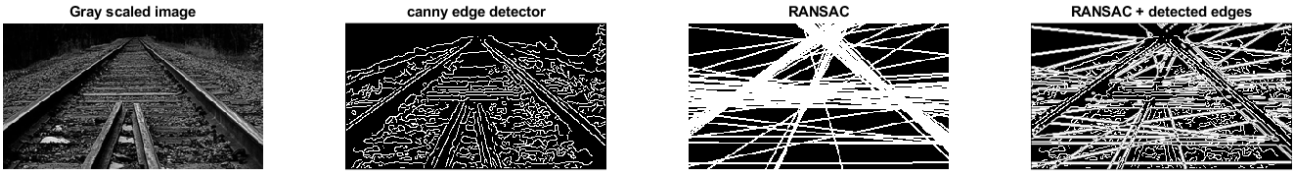


Figure 26 Important lines in the railroad image $\Delta r = 1, \Delta\theta = 1, Cth = 5$

The same strategy could be done by changing the value of threshold; for r, if we increase Δr, more lines are found.

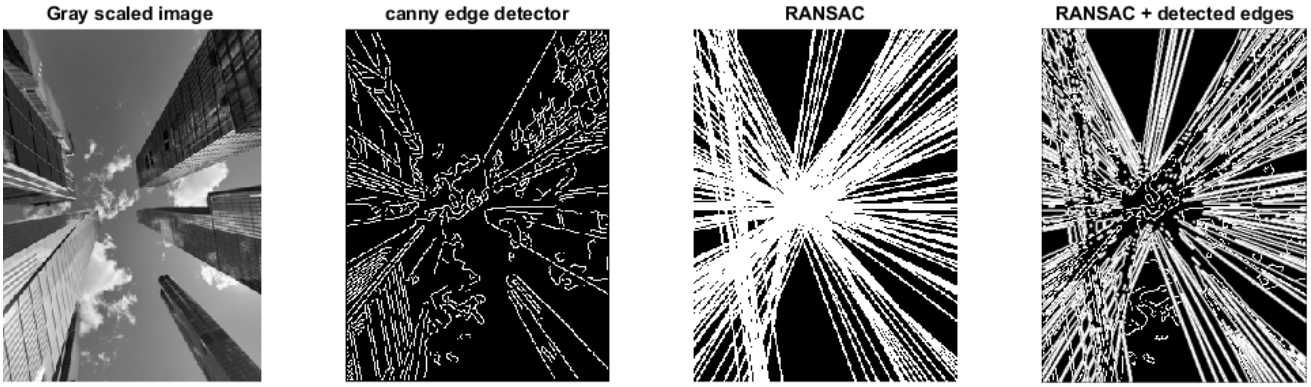


Figure 27 Important lines in the skyscraper image $\Delta r = 3, \Delta\theta = 1, Cth = 10$

Figure 28 Important lines in the railroad image $\Delta r = 3, \Delta\theta = 1, Cth = 10$

PART B

[*Attempt this part only once you have a solution to part a), since this part carries only a little weight*]. Modify your solution in part a) slightly as follows. Rather than using the average position and orientation to return a line model, produce one via a least squares fit to the (x, y) data points in the consensus set. To implement least squares fitting you can use built in Matlab functions for eigenvector/eigenvalue computation. The difference now will be that your line models will involve actual fits to the data points, and each line model will come with an error of fit. Discuss your results.

ANSWER – PART B

In this part of the question, the algorithm is somehow the same; however, in the previous strategy, we just calculated a sample mean among the found in-line lines to find a single significant line. But now, this single significant line is found by using a total least square fitting. In this new strategy, if we randomly chose an edge and then find a number of edges in line with that (let us say the total number of edges in this cluster is C), we will have C data points each correspond to a specific r and θ value. We want to find a line that could cause the minimum error of fitting. In 2D, for a single cluster, we could have such an illustration:

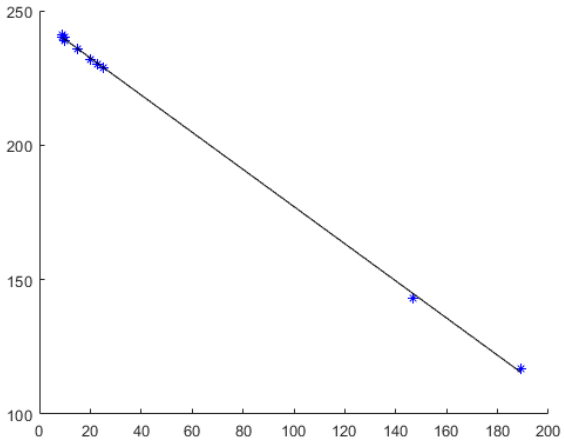


Figure 29 A sample line fitting by using the total least square strategy

Each of these blue dots corresponds to an edge within the same cluster, and here, we can see the fitted line using total least squares. But the question is: how can we obtain such a line? To find this fitted line, we first make a $A^T A$ which is a 2 in 2 matrix as below:

$$\begin{bmatrix} \sum_{i=1}^{N} (x_i - \bar{x})^2 & \sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y}) \\ \sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y}) & \sum_{i=1}^{N} (y_i - \bar{y})^2 \end{bmatrix}$$

and then try to find θ as it can minimize the expression below:

$$[\cos\theta, \sin\theta]\ \boldsymbol{A^T A}\begin{bmatrix}\cos\theta \\ \sin\theta\end{bmatrix}$$

To find such optimum value of θ we need to find the eigen values and eigen vectors of the $A^T A$, and we then take the eigenvector which has the smallest eigenvalue and choose θ in a way that it $[cos\theta, \sin\theta]T$ is this eigenvector. The same procedure in done for each of the found clusters and you can see the ultimate result in Figure 30.

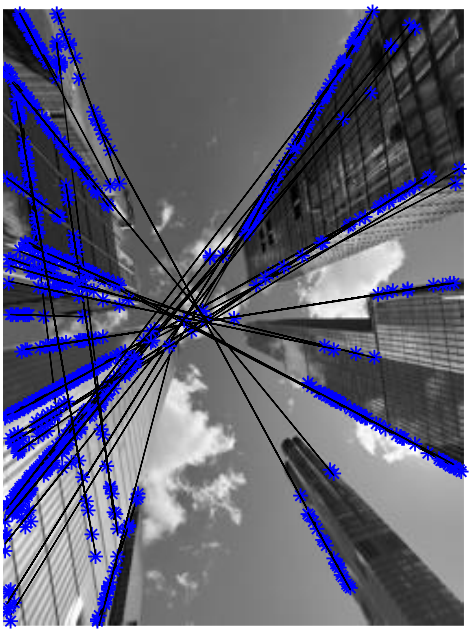


Figure 30 Important lines in the skyscraper image $\boldsymbol{\Delta r = 1, \Delta\theta = 1, Cth = 10}$

total least square

Figure 31 Important lines in the railroad image $\boldsymbol{\Delta r = 1, \Delta\theta = 1, Cth = 10}$

total least square

REFRENCES

[1] Marr, David, and Ellen Hildreth. "Theory of edge detection." *Proceedings of the Royal Society of London. Series B. Biological Sciences* 207.1167 (1980): 187-217.

[2] Spontón, Haldo, and Juan Cardelino. "A review of classic edge detectors." *Image Processing On Line* 5 (2015): 90-123.

[3] Shubham, Saini, Kasliwal Bhavesh, and Bhatia Shraey. "Comparativestudy of imageedge detection algorithms." *arXiv preprint arXiv: 1311* 496 (2013).

[4] Kumar, Mukesh, and Rohini Saxena. "Algorithm and technique on various edge detection: A survey." *Signal & Image Processing* 4.3 (2013): 65.