

Numerical Analysis

Computational Project

CS 2 - Noe Lomidze

2.1

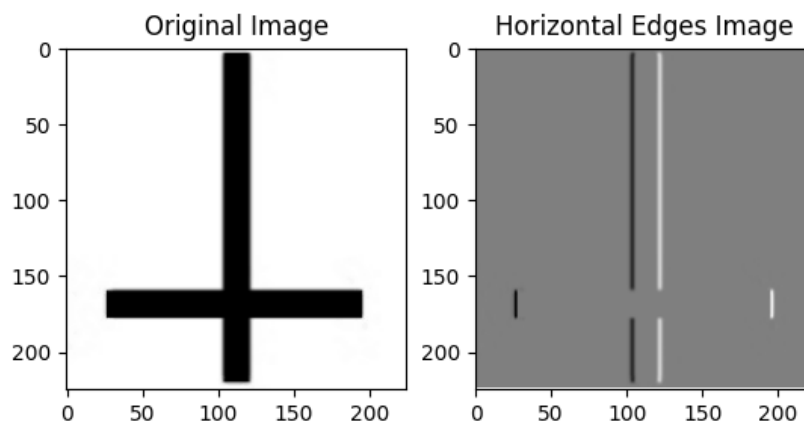
Problem 2.1

Digital Images, Edges, and Derivatives (3 points)

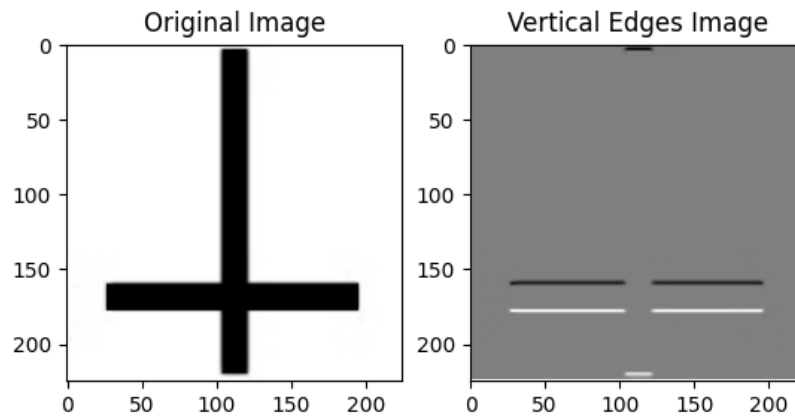
1. Define edges and provide examples to illustrate the concept.
2. Explain how derivatives are utilized for edge detection in one and two dimensions. Describe edge indicators and supplement your explanation with visual examples.
3. Investigate the impact of truncation error in finite difference formulas on edge detection. Support your findings with visual evidence.

Please use a separate slide for each of the sub-tasks mentioned above.

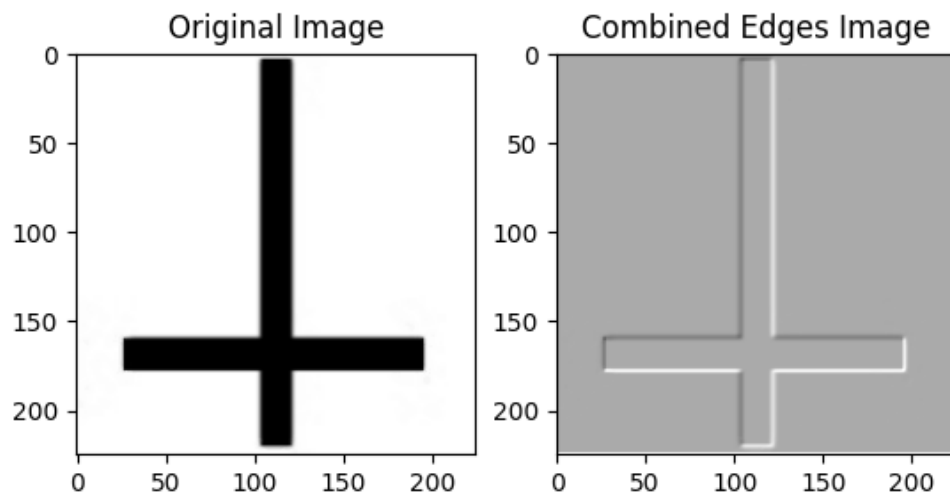
So firstly, I did some edge detection on black and white photo, with [Sobel](#) matrix and this was the result. Here I used the [horizontal](#) filter matrix.



Then I used the [vertical](#) filter matrix, which is just a [transpose](#) of the horizontal matrix.



Then I used [both](#) filters together and got the result like that



This is what the process looks like:

Original Image:

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

Vertical Filter:

-1	-2	-1
0	0	0
1	2	1

Apply Filter to Pixels:

-1	-2	-1
0	0	0
0	0	0

Sum = **-4** which is Min Value, map to **0**

Copyright to ritvikmath for those photos

Original Image:

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

Vertical Filter:

-1	-2	-1
0	0	0
1	2	1

Apply Filter to Pixels:

0	0	0
0	0	0
1	2	1

Sum = **4** which is Max Value, map to **1**

Here are two beautiful examples:

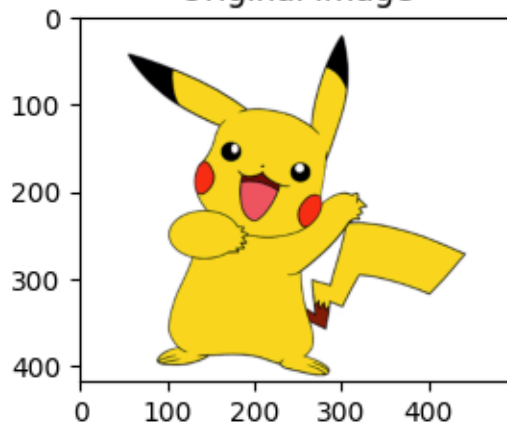
Original Image



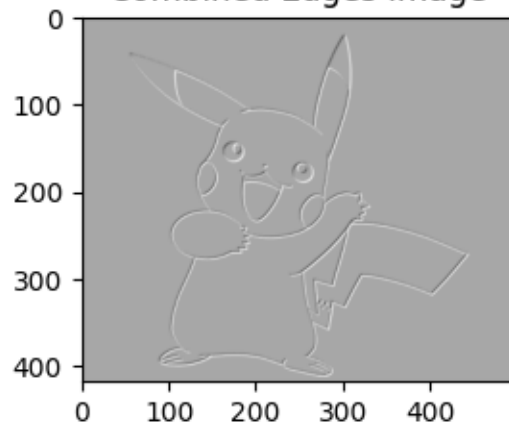
Edges Detected



Original Image



Combined Edges Image

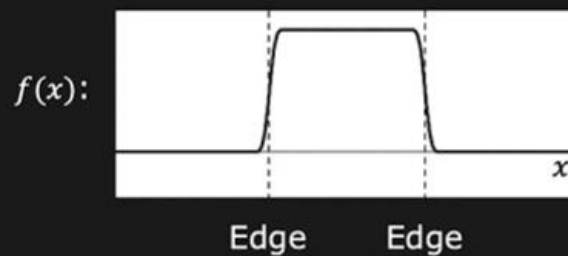


How are [derivatives](#) utilized in edge detection?

1D edge detection:

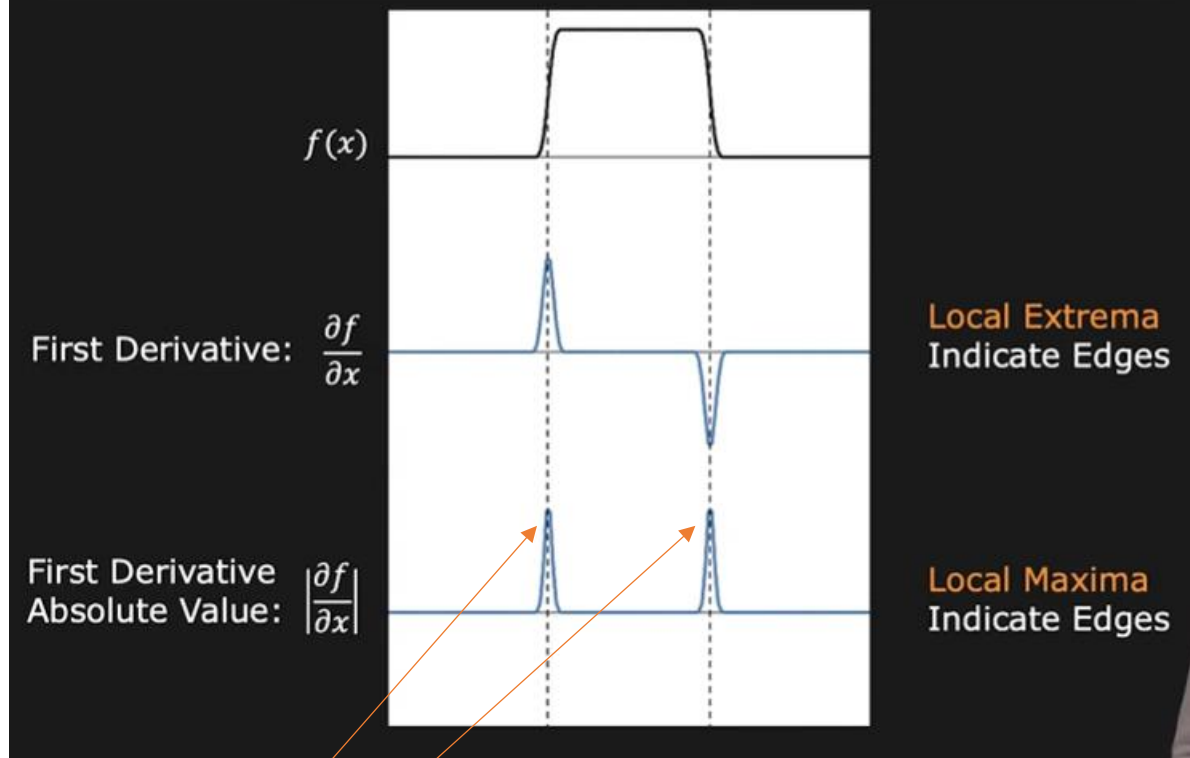
1D Edge Detection

Edge is a rapid change in image intensity in a small region.



Basic Calculus: **Derivative** of a continuous function represents the amount of change in the function.

Edge Detection Using 1st Derivative



Therefore, as you can see, the local extrema in the first derivative tell you where the edges are, and all you need to do is to find the absolute of the output, where you get those two peaks, as in the picture. The locations of the peaks tell you where the edges are and the height of the peak should tell you what the strength of the edge is.

Now, we would like to apply this idea of using first derivatives to [two-dimensional images](#). So what do we do in case of two-dimensional images?



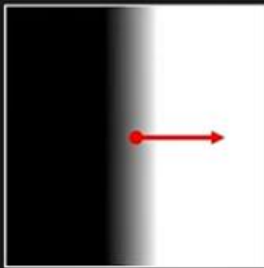
Basic Calculus: [Partial Derivatives](#) of a 2D continuous function represents the amount of change along the each dimension.

Gradient (∇)

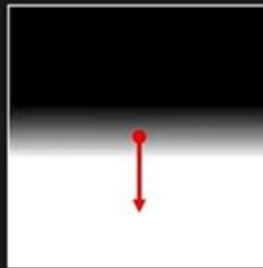
Gradient (Partial Derivatives) represents the direction of most rapid change in intensity

$$\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right]$$

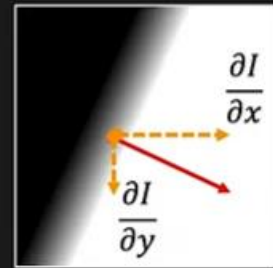
Pronounced as "Del I"



$$\nabla I = \left[\frac{\partial I}{\partial x}, 0 \right]$$



$$\nabla I = \left[0, \frac{\partial I}{\partial y} \right]$$

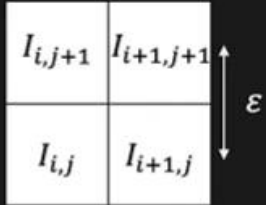


$$\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right]$$

Therefore, we are talking about derivatives here, and we already know from the course of [Numerical Analysis](#) that derivatives are implemented using [finite differences](#), especially when it comes to discrete images:

Finite difference approximations:

$$\frac{\partial I}{\partial x} \approx \frac{1}{2\varepsilon} \left((I_{i+1,j+1} - I_{i,j+1}) + (I_{i+1,j} - I_{i,j}) \right)$$

$$\frac{\partial I}{\partial y} \approx \frac{1}{2\varepsilon} \left((I_{i+1,j+1} - I_{i+1,j}) + (I_{i,j+1} - I_{i,j}) \right)$$


Copyright to First Principles of Computer Vision

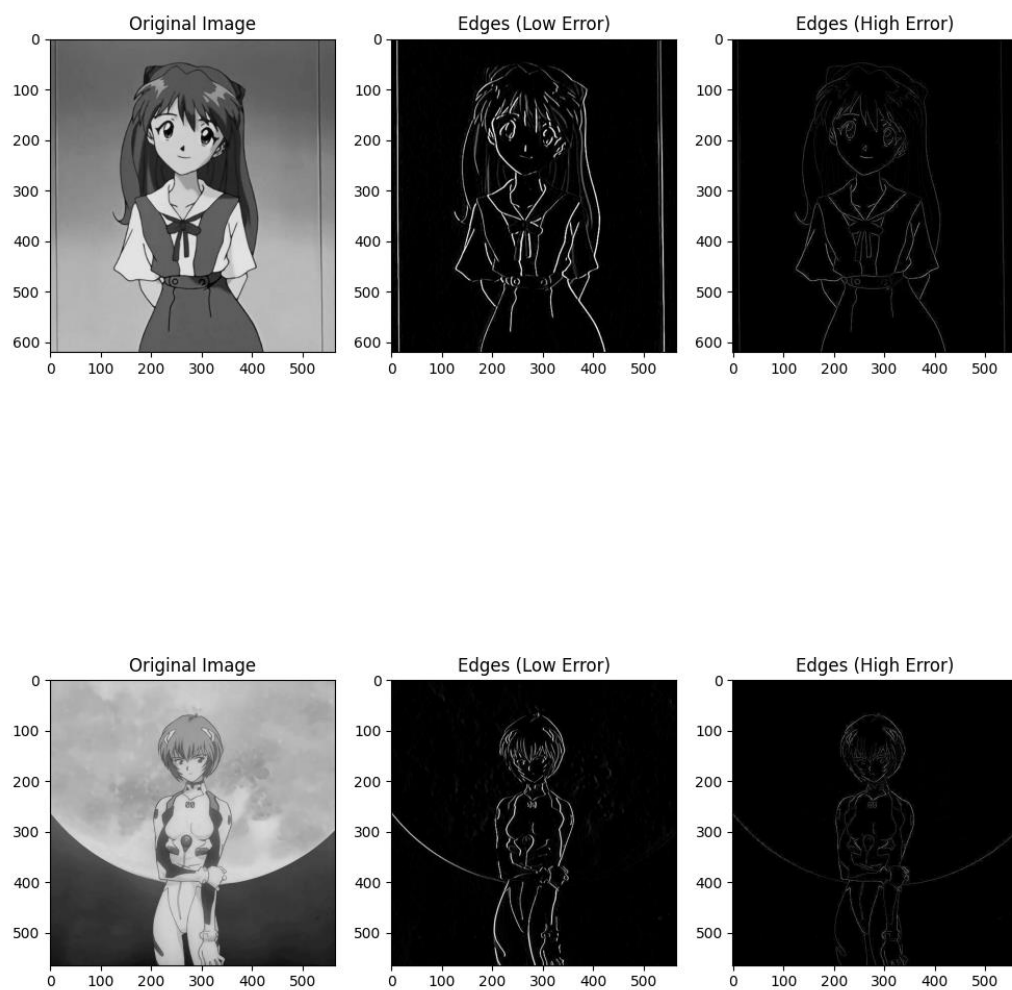
So now, let us compare different gradient operators.

[Sobel 3x3 is the one I used in the beginning](#)

Comparing Gradient (∇) Operators

Gradient	Roberts	Prewitt	Sobel (3x3)	Sobel (5x5)																																															
$\frac{\partial I}{\partial x}$	<table><tr><td>0</td><td>1</td></tr><tr><td>-1</td><td>0</td></tr></table>	0	1	-1	0	<table><tr><td>-1</td><td>0</td><td>1</td></tr><tr><td>-1</td><td>0</td><td>1</td></tr><tr><td>-1</td><td>0</td><td>1</td></tr></table>	-1	0	1	-1	0	1	-1	0	1	<table><tr><td>-1</td><td>0</td><td>1</td></tr><tr><td>-2</td><td>0</td><td>2</td></tr><tr><td>-1</td><td>0</td><td>1</td></tr></table>	-1	0	1	-2	0	2	-1	0	1	<table><tr><td>-1</td><td>-2</td><td>0</td><td>2</td><td>1</td></tr><tr><td>-2</td><td>-3</td><td>0</td><td>3</td><td>2</td></tr><tr><td>-3</td><td>-5</td><td>0</td><td>5</td><td>3</td></tr><tr><td>-2</td><td>-3</td><td>0</td><td>3</td><td>2</td></tr><tr><td>-1</td><td>-2</td><td>0</td><td>2</td><td>1</td></tr></table>	-1	-2	0	2	1	-2	-3	0	3	2	-3	-5	0	5	3	-2	-3	0	3	2	-1	-2	0	2	1
0	1																																																		
-1	0																																																		
-1	0	1																																																	
-1	0	1																																																	
-1	0	1																																																	
-1	0	1																																																	
-2	0	2																																																	
-1	0	1																																																	
-1	-2	0	2	1																																															
-2	-3	0	3	2																																															
-3	-5	0	5	3																																															
-2	-3	0	3	2																																															
-1	-2	0	2	1																																															
$\frac{\partial I}{\partial y}$	<table><tr><td>1</td><td>0</td></tr><tr><td>0</td><td>-1</td></tr></table>	1	0	0	-1	<table><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>-1</td><td>-1</td><td>-1</td></tr></table>	1	1	1	0	0	0	-1	-1	-1	<table><tr><td>1</td><td>2</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>-1</td><td>-2</td><td>-1</td></tr></table>	1	2	1	0	0	0	-1	-2	-1	<table><tr><td>1</td><td>2</td><td>3</td><td>2</td><td>1</td></tr><tr><td>2</td><td>3</td><td>5</td><td>3</td><td>2</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>-2</td><td>-3</td><td>-5</td><td>-3</td><td>-2</td></tr><tr><td>-1</td><td>-2</td><td>-3</td><td>-2</td><td>-1</td></tr></table>	1	2	3	2	1	2	3	5	3	2	0	0	0	0	0	-2	-3	-5	-3	-2	-1	-2	-3	-2	-1
1	0																																																		
0	-1																																																		
1	1	1																																																	
0	0	0																																																	
-1	-1	-1																																																	
1	2	1																																																	
0	0	0																																																	
-1	-2	-1																																																	
1	2	3	2	1																																															
2	3	5	3	2																																															
0	0	0	0	0																																															
-2	-3	-5	-3	-2																																															
-1	-2	-3	-2	-1																																															

As you can see below, the **impact of truncation** error in finite difference formulas is **crucial** on edge detection.



Problem 2.2

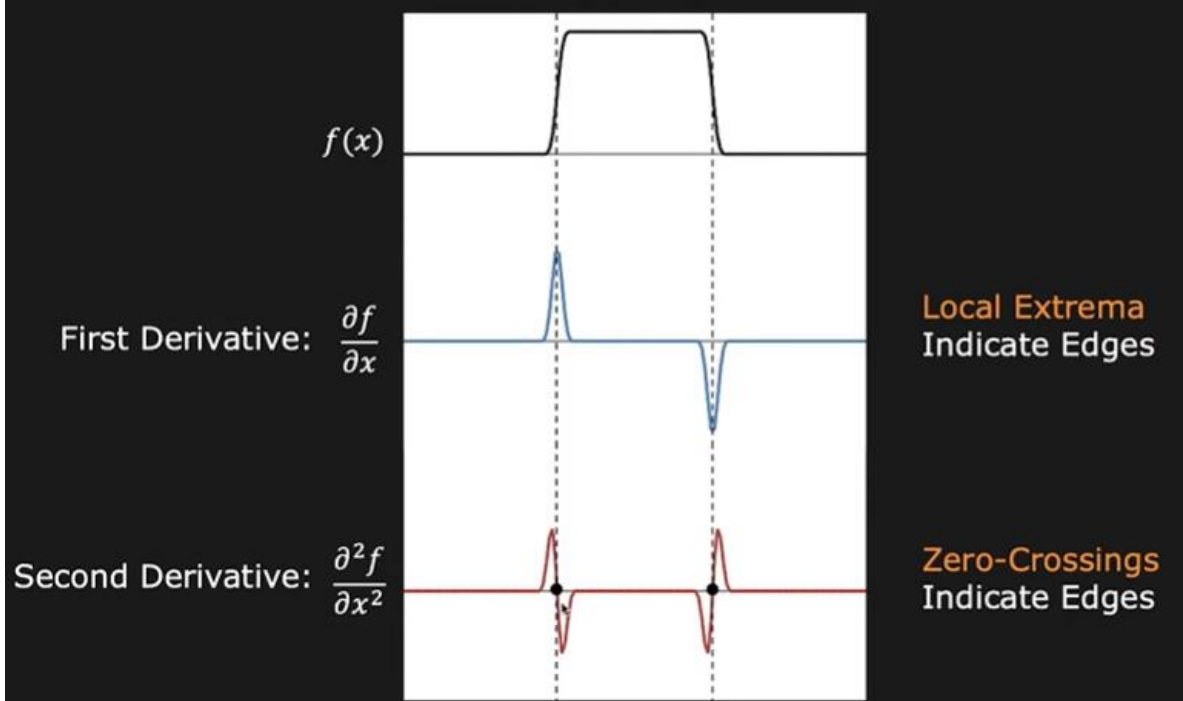
Digital Images, Features, and Higher Order Derivatives (3 points)

1. Investigate whether higher order derivatives can be employed for edge detection.
2. Explore the potential of higher order derivatives in extracting other features of digital images.
3. Present visual examples to illustrate both successful and unsuccessful findings.

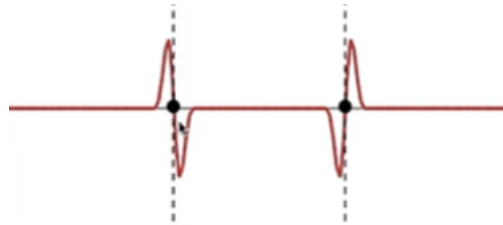
Please use a separate slide for each of the sub-tasks mentioned above.

Now let us look at the use of [second derivative](#) on edge detection

Edge Detection Using 2nd Derivative



If you look at the picture, the first derivative behaves pretty well, it's zero on edges, but if you look at the second derivative, it gets more and more interesting.



Prediction would be that, the second derivative would be zero, where the first derivative had extremums, and so on and so on.

As you can see, you get a very sharp change from positive to negative, and this is called [Zero Crossing](#).

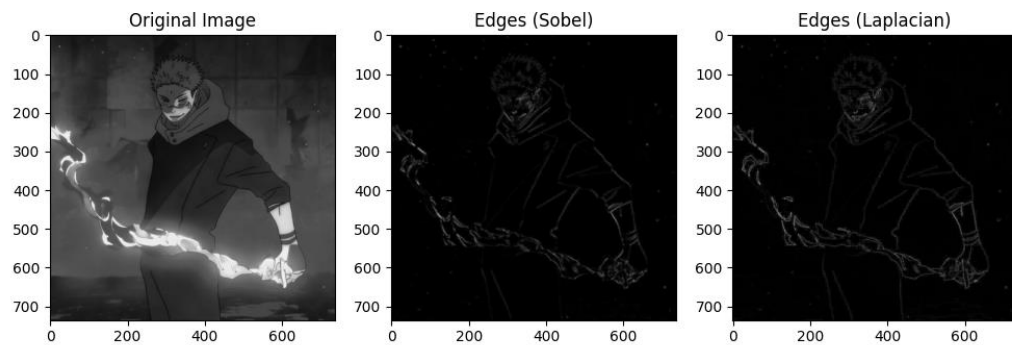
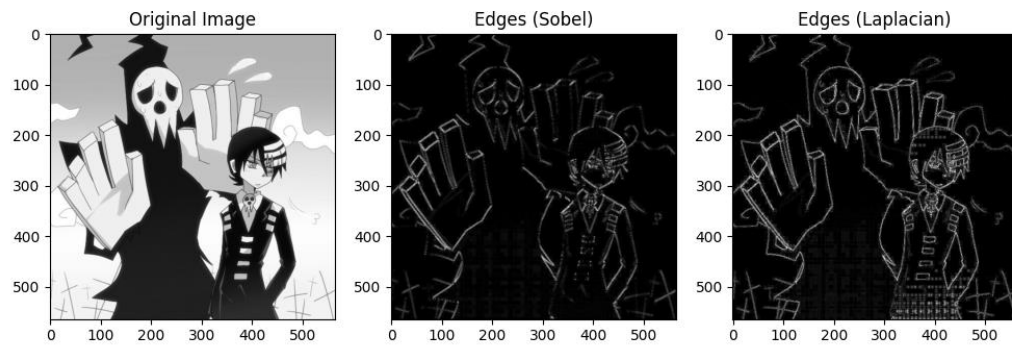
To sum up, what is interesting about the second derivative of f with respect to x is that, at the edges, you do not get peaks, you in fact get zeros, but you get very strong zero-crossings. And most importantly, if you can detect those zero-crossings, wherever there is a zero-crossing, you have an edge.

Moreover, that brings us to the [Laplacian operator](#), which is familiar for us, thanks to Analysis 2.

$$\nabla \cdot \nabla f = \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Therefore, when you apply the Laplacian operator to the image, you are going to end up with zero-crossings, and that is where the edges lie.

Now let us compare the Sobel, which uses the first derivatives and the Laplacian, which as you see, uses the second derivatives.



The distinction is obvious in the first instance, but in the second one, it is not as apparent as before.

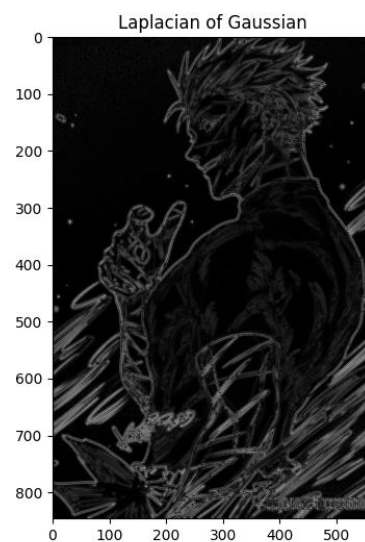
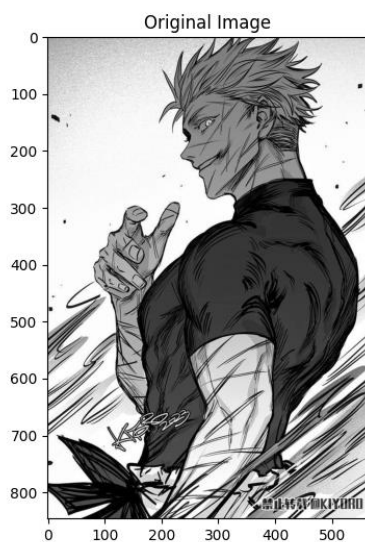
Now about the higher order derivatives in edge detection

I did some research and found some interesting things where higher order derivatives are useful, for example:

Texture Analysis:

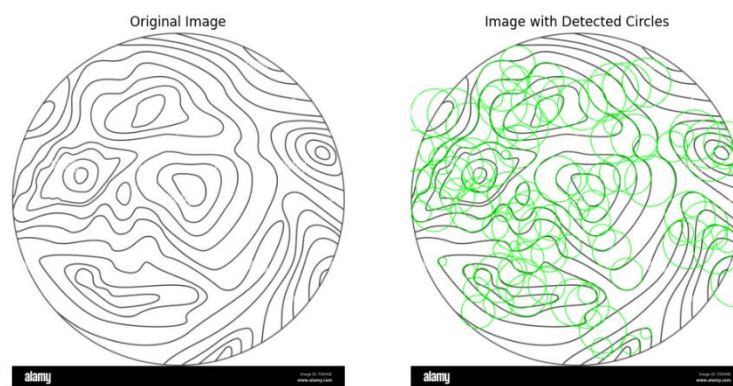
Higher order derivatives can be utilized for texture analysis, where the goal is to **characterize the spatial arrangement of pixels in an image**. You can apply filters based on higher order derivatives to highlight textures with specific properties, such as **roughness or regularity**. For instance, the second-order derivatives can be used to compute the local structure tensor, which provides information about the orientation and anisotropy of texture patterns.

Now focus on texture analysis using higher order derivatives. I will use the Laplacian of Gaussian (LoG) operator to enhance textures in an image:

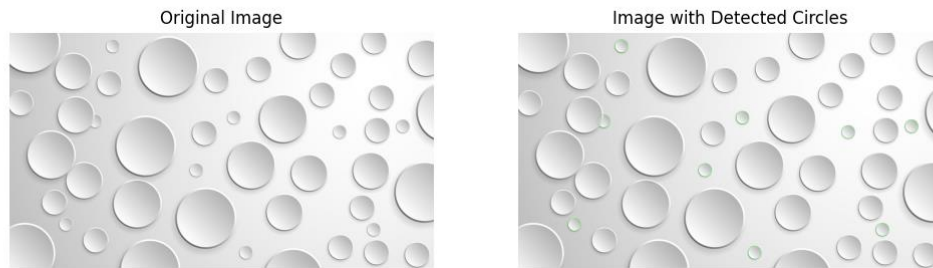


Another example:

Shape Detection: Higher order derivatives can help in detecting shapes and contours in an image. By analyzing the **curvature** and higher-order derivatives of intensity profiles along contours, you can **identify regions** corresponding to specific shapes or objects. For example, you can use the curvature information extracted from higher order derivatives to detect circular or elliptical shapes in the image.



In this example, I am attempting to detect **circular shapes** in an image with a complex background using the **Hough Circle Transform**. Since the background introduces noise and clutter, the algorithm may fail to accurately detect the circles, leading to unsuccessful findings as you can see below



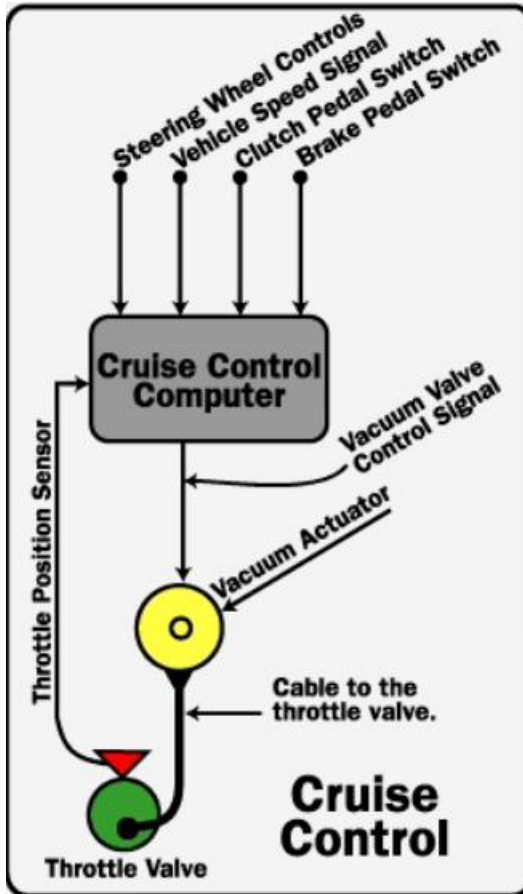
Problem 2.3

Exploring the Applications of Derivatives (3 points)

1. Provide two illustrations of how derivatives are applied in real-world scenarios.
2. Ensure one of the examples is connected with the concept of edge detection.
3. Present a visual representation for at least one of the examples. Clarify why this application works and mention the tools that can be utilized for this purpose.

Please use a separate slide for each of the sub-tasks mentioned above.

Cruise control systems maintain constant vehicle speed without manual throttle adjustments, utilizing derivatives for control.



Derivative Measurement: The cruise control system continuously monitors the vehicle's speed, often using sensors that measure wheel rotation or engine RPM. By measuring how quickly the vehicle's speed is changing over time, the system obtains the **derivative of the speed function, which represents the rate of change of speed**.

Feedback Control: Derivative feedback in a closed-loop system compares actual and desired speeds, minimizing deviations for smooth driving.

Smooth Acceleration: Incorporating derivative feedback ensures smooth acceleration and deceleration, enhancing passenger comfort.

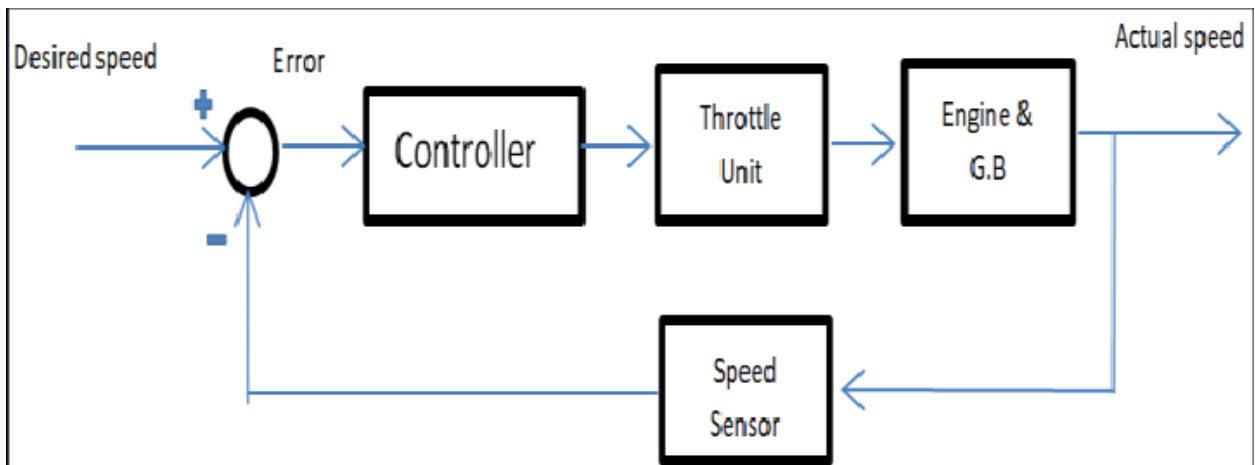
Safety and Efficiency: Cruise control systems improve safety and fuel efficiency by preventing sudden speed changes, aided by derivative-based control algorithms.

When would you use cruise control?

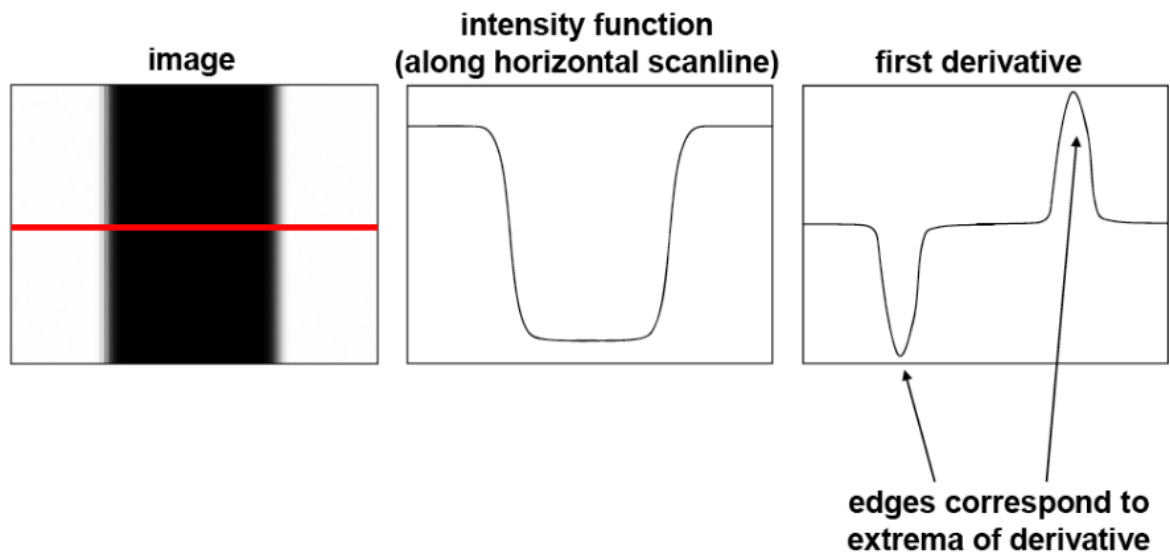
Cruise control is best used on long drives where the speed limit remains the same over much of the distance. It is great for low to no traffic situations where you do not need to change or adjust speed often.

How useful is cruise control?

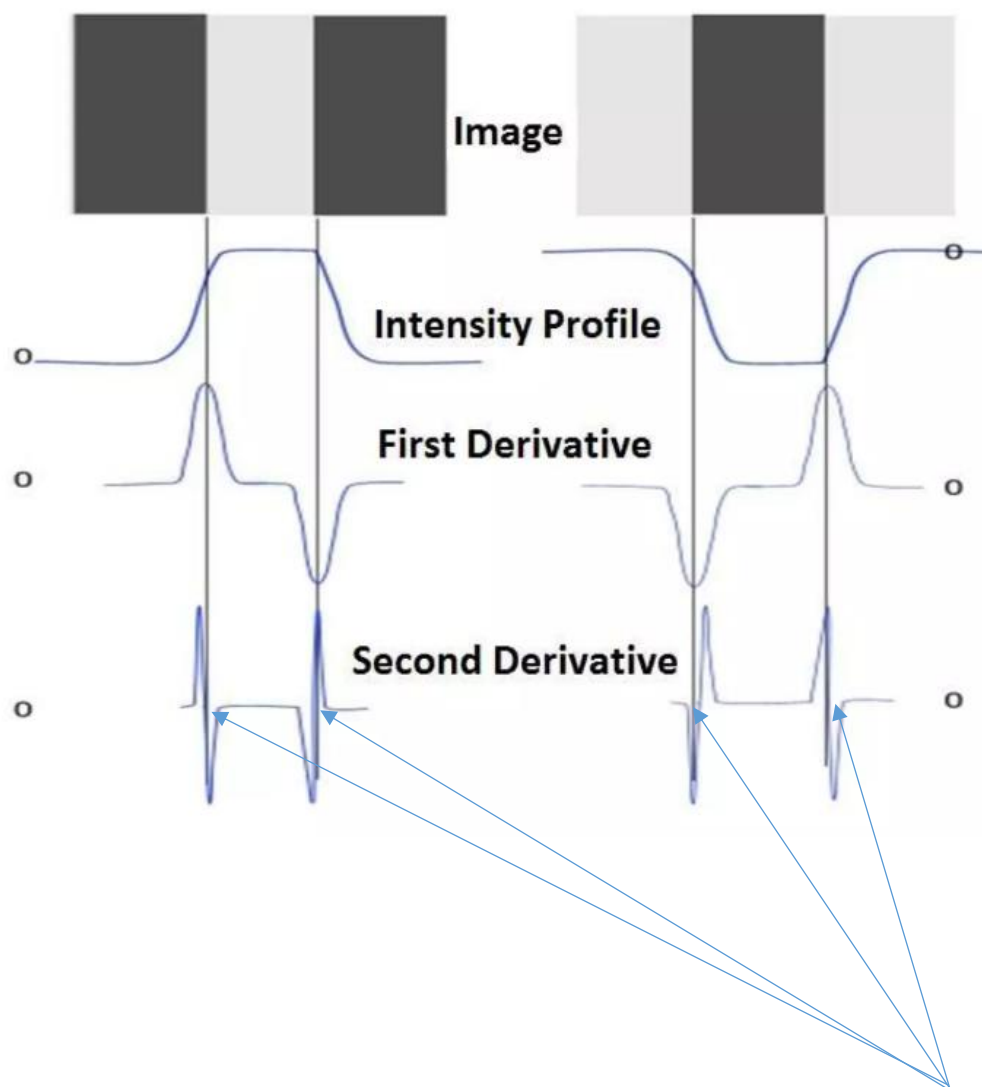
Cruise control can reduce a driver's fatigue and improve comfort while driving. It can also help drivers stay within the speed limit.



So now, let us talk about edge detection again.



Edge detection by derivatives

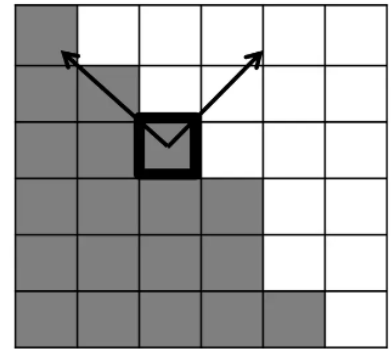


Derivatives are great tool to use in edge detection as I mentioned before and this picture exactly describes that. We can talk about the image by its Intensity profile function derivatives, in the first derivative, edges are at the peaks, extrema values, and the second derivative provides a better identification of edges by zero-crossings.

Edge detection by first derivative

- $I = f(x, y)$

- $\vec{\nabla} f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$

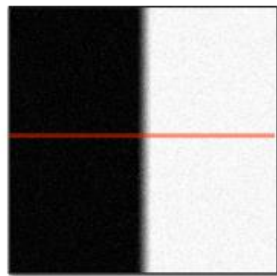


- $\nabla f = mag(\vec{\nabla} f) = \sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y|$

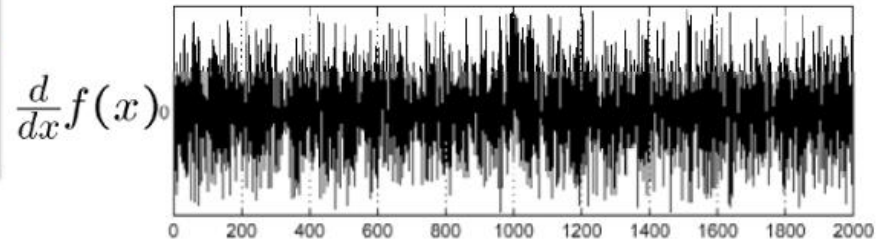
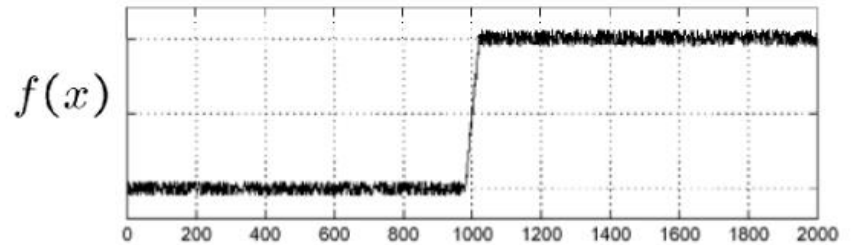
- Direction of $\vec{\nabla} f$, $\alpha(x, y) = \tan^{-1} \left(\frac{G_y}{G_x} \right)$

So then, comes the question, what if our image is noisy? What can we do about it?

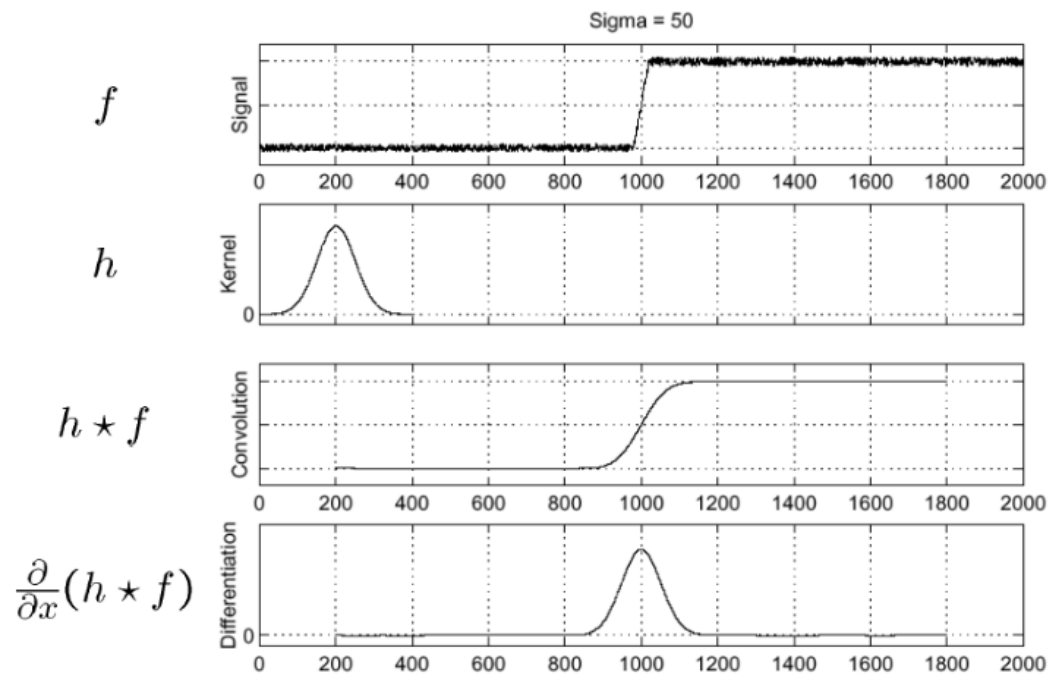
Plotting intensity as a function of position gives a signal.



Noisy input image



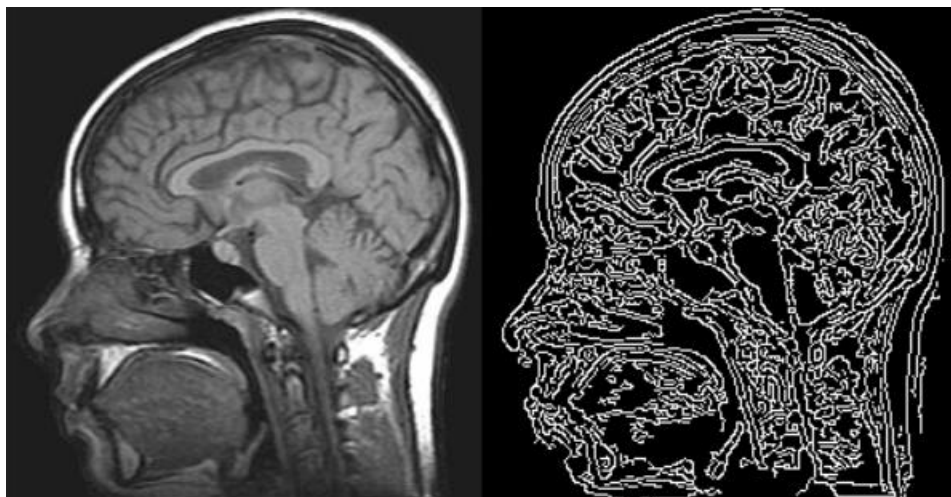
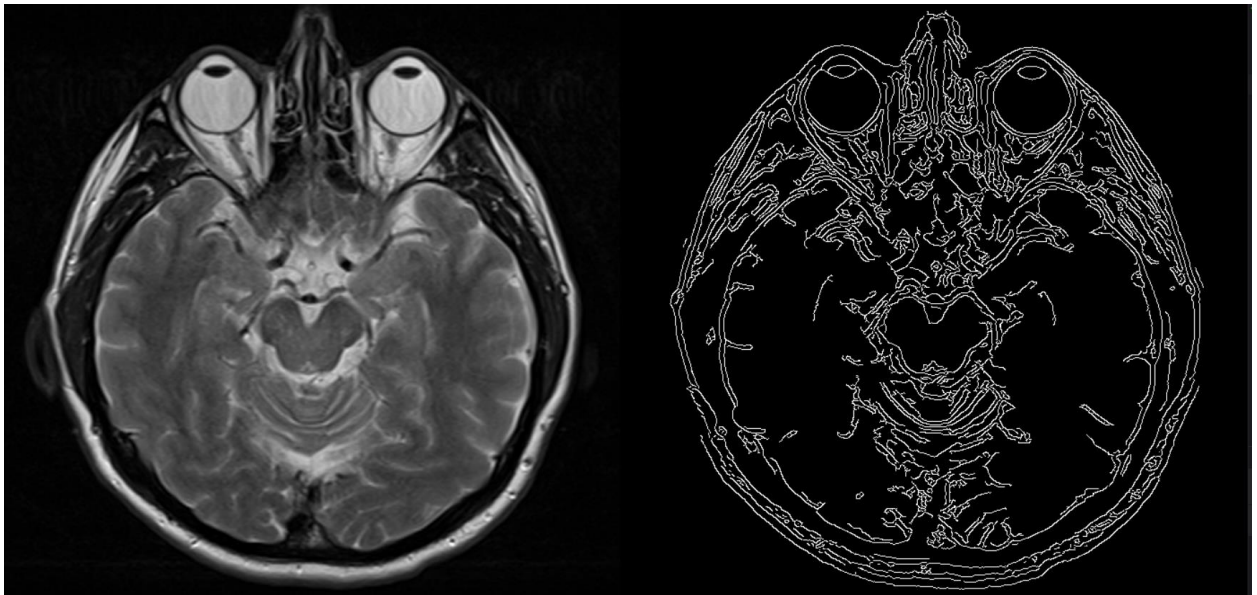
Smooth first with h (e.g. Gaussian), and look for peaks in $\frac{\partial}{\partial x}(h * f)$.



Problem Solved

In neuroimaging, [edge detection](#) is crucial for analyzing brain images from techniques like MRI.

It helps identify brain structures, reconstruct neuronal morphology, map brain function, detect tumors, and diagnose neurological disorders.



Problem 2.4

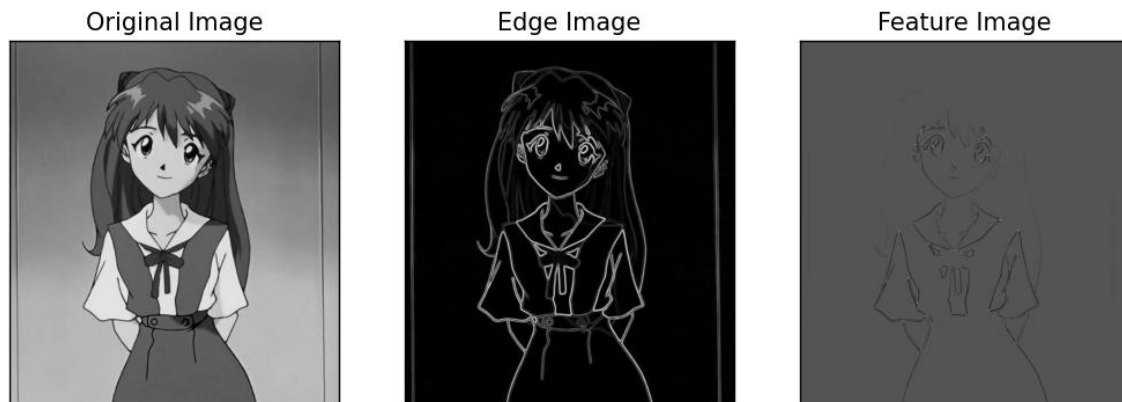
Digital Images, Features, and Linear Combinations of Derivatives (3 points)

1. Investigate whether linear combinations of derivatives can be employed for edge detection. Describe the method used to select coefficients in the linear combination of derivatives and justify your approach.
2. Explore the potential of linear combinations of derivatives in extracting other features of digital images. Describe the method used to select coefficients in the linear combination of derivatives and justify your approach.
3. Present visual examples to illustrate both successful and unsuccessful findings.

I did a little research and this is what I got.

To form a linear combination of derivatives for edge detection, you could use the first order derivatives in the x and y directions (Sobel operators), and combine them. The coefficients in the linear combination could be selected based on the importance you want to give to changes in the respective directions. For instance, if you want to give equal importance to changes in both directions, you could use equal coefficients. (We also learnt throughout the NA course that Method of undetermined coefficients might be useful)

The justification for this approach is that edges correspond to high intensity changes, and derivatives capture these changes. By adjusting the coefficients, you can control the sensitivity to changes in different directions.



Edge Detection Using Linear Combinations of Derivatives:

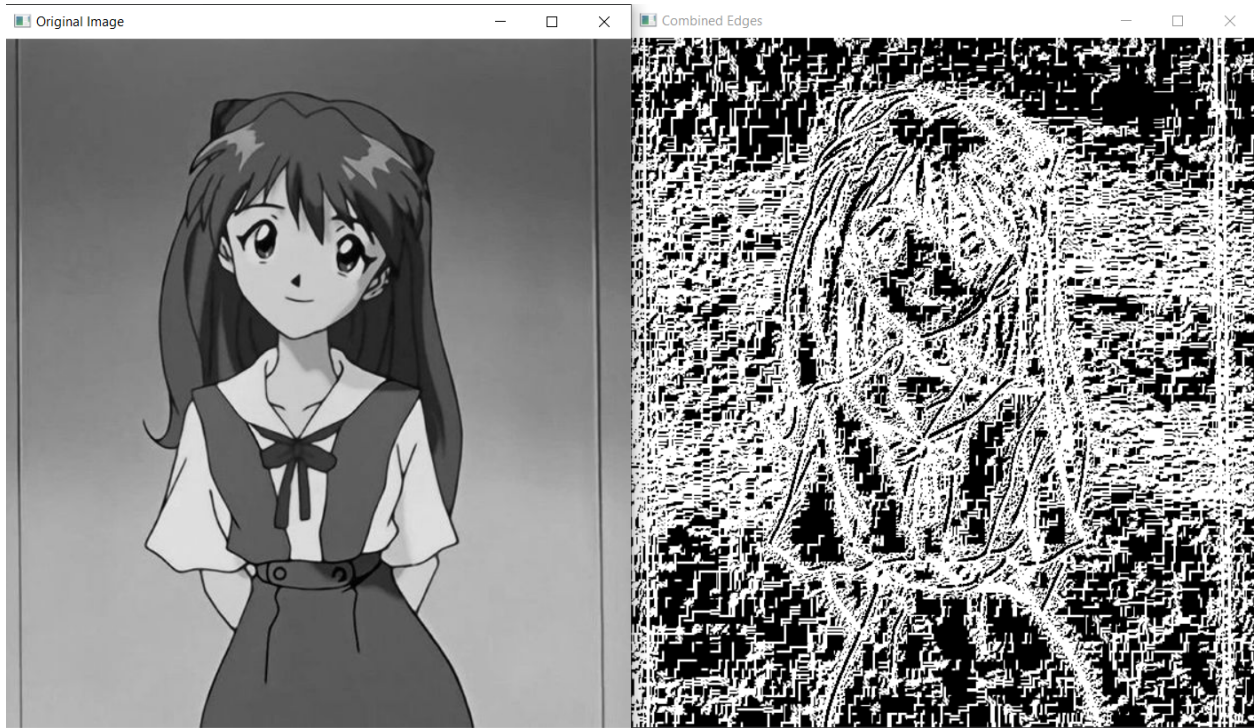
The Sobel operator is used in the code for edge detection. It computes the gradient of the image intensity at each pixel and represents the image as a collection of edges. It does this by calculating the first order derivatives in the x (sobelx) and y (sobely) directions.

The `cv2.Sobel()` function is used to calculate these derivatives. The linear combination of these derivatives is then calculated using the `np.hypot()` function, which computes the square root of the sum of squares of its inputs. This effectively gives us the magnitude of the gradient at each pixel, highlighting the edges. The coefficients in the linear combination are implicitly set to 1 by the `np.hypot()` function, giving equal importance to changes in both the x and y directions. This is justified as we generally want to detect edges in all directions.

Extracting Other Features Using Linear Combinations of Derivatives:

The Harris corner detector is used in the code for feature extraction. It operates in a similar way to the Sobel operator, but instead of looking for high gradient regions, it looks for regions with large changes in all directions. This is achieved by considering the second order derivatives in both directions. The `cv2.cornerHarris()` function is used to perform this operation.

The coefficients in the linear combination for the Harris corner detector are determined internally by the algorithm. It is designed to give more importance to regions where both the second order derivatives are large, which corresponds to corners.

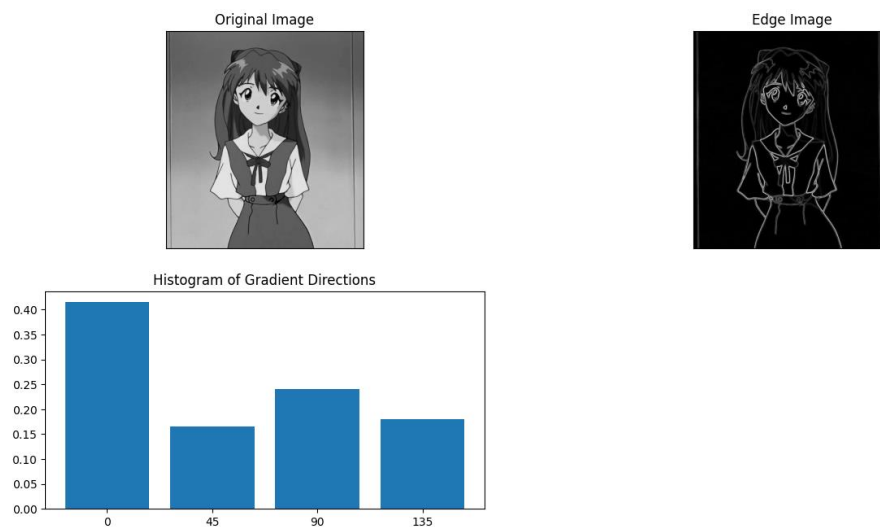


We can consider this as an unsuccessful example while combining the edges, in my python file Problem 2.4/Test1.py

Edge Detection using Linear Combinations of Derivatives:

Edge detection in images is a crucial aspect of computer vision and is used in a variety of applications such as image segmentation, object detection, etc. One common method for edge detection is the use of derivatives. The idea is that the intensity of an image changes rapidly at the edges, and these changes can be captured by derivatives.

A linear combination of derivatives can be used to detect edges in multiple directions.



Other features of digital images can also be extracted using linear combinations of derivatives. For example, texture features can be extracted by computing the gradient of the image and then quantizing the result into a histogram as you can see in the picture