CO543 Image Processing
Lab 06
Feature Extraction

Goal

In this lab, we aim to understand what features are, why they are important in image processing, and how corners play a crucial role in various computer vision tasks. We will explore different feature detection algorithms and learn how to implement them using OpenCV.

Introduction

Imagine playing a jigsaw puzzle. You have many small pieces that need to be assembled to form a complete picture. The key is identifying unique, trackable, and comparable features in images. These features help align and stitch images together. Humans can naturally identify these features, but for computers, we need to define them algorithmically.

For example, consider the following image:

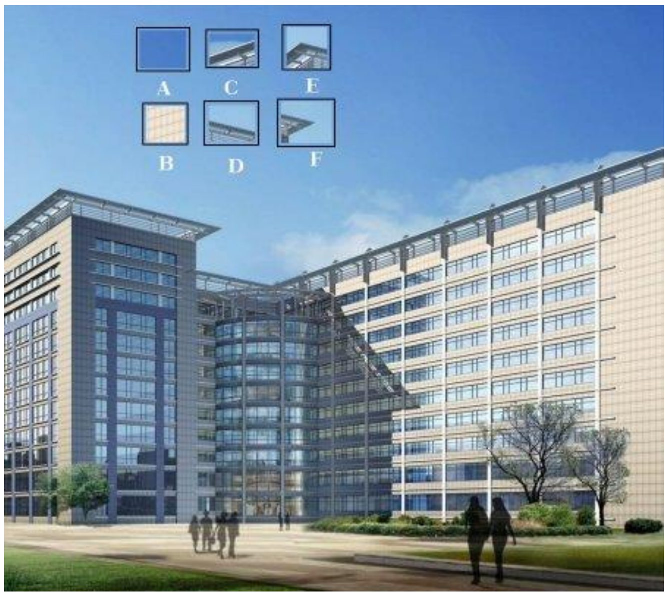

Figure 1: Image of a building. (Source)

At the top of the image, there are six small patches. If you were asked to find their exact locations in the original image, how would you proceed?

- **A and B** are flat surfaces spread over a large area, making it difficult to pinpoint their exact locations.
- **C and D** are edges of the building, allowing for approximate localization but not exact, as the pattern repeats along the edge.
- **E and F** are corners, easily identifiable since moving these patches will always produce a different look, making them unique.

Therefore, we can see that edges and corners are two primary features which can be used to distinguish image patches.

Part 1: Edge Detection

Edges represent the boundaries between different regions in an image. They are essential for understanding the structure of objects within the image. To detect edges, we use derivatives to highlight regions where there is a significant change in intensity. Various algorithms help us identify these changes effectively.

1. Sobel Operator:
   The Sobel operator uses first-order derivatives to obtain pixel intensity ***changes*** in an image.
   For example, we can detect horizontal changes using the G_x 3x3 kernel given below and vertical changes using the 3x3 kernel G_y.

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

G_x

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

G_y

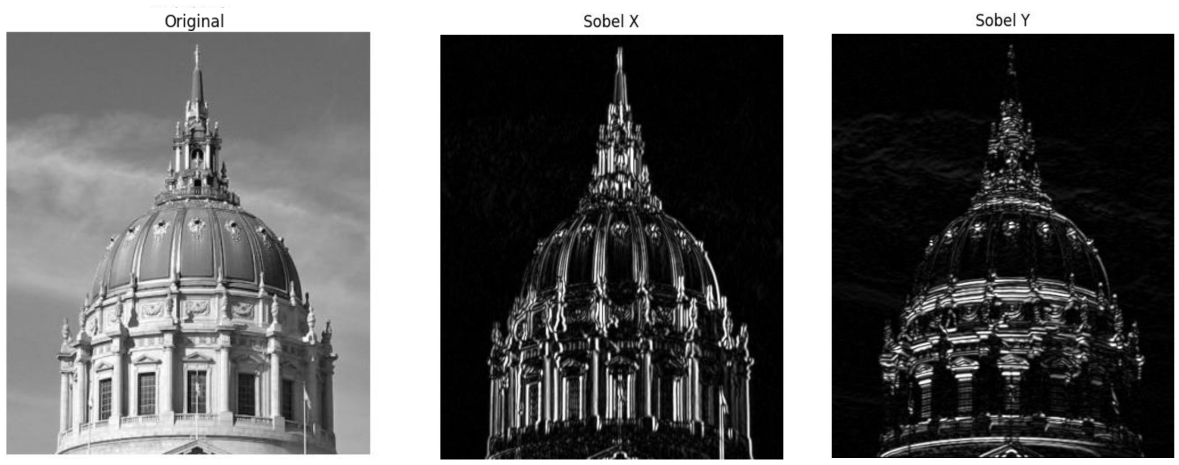

Figure 2: Vertical and horizontal edges detected by Sobel Operators

2. Laplacian Operator:
   The Laplacian operator uses the second derivative to ***detect edges*** by taking derivates in both dimensions in a 2D image.

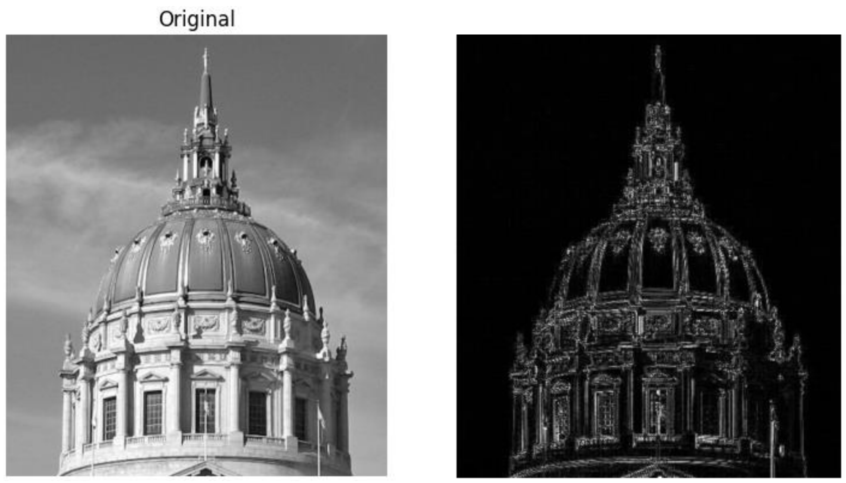

Figure 2: Edges detected by Laplacian Operator

3. Canny Edge Detector:
   The Canny edge detector is a multi-stage algorithm developed by John F. Canny in 1986, that uses gradient-based edge detection with noise reduction and edge tracing. It is widely used for its accuracy and performance.

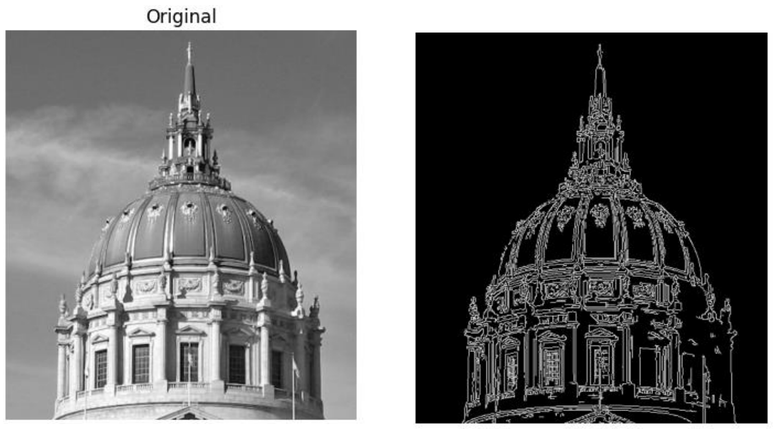

Figure 3: Edges detected by Canny Edge Detector

4. Hough Line Transform:

   The Hough Transform is used to detect lines in an image by transforming points in the image space to the parameter space and identifying intersecting points that represent lines. To apply the Transform, first an edge detection pre-processing such as Canny Edge Detection is desirable.

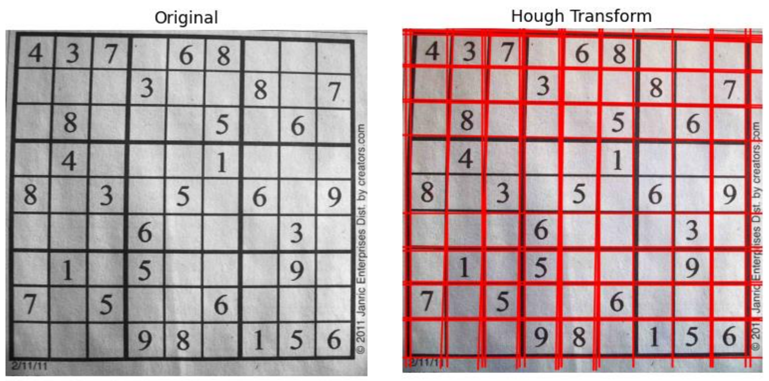

Figure 4: Lines detected by Hough Line Transform

Part 2: Corner Detection

Corners are points in an image where the intensity changes sharply in multiple directions. They are critical for tasks such as object recognition, image stitching, and 3D reconstruction because they are unique and can be easily tracked across different images.

1. Harris Corner Detection:
   The Harris Corner Detector is an algorithm that detects corners by looking for significant changes in intensity in multiple directions.
2. Shi-Tomasi Corner Detector:
   The Shi-Tomasi Corner Detector improves on the Harris Detector by considering only the minimum eigenvalue, making it more accurate for certain applications.

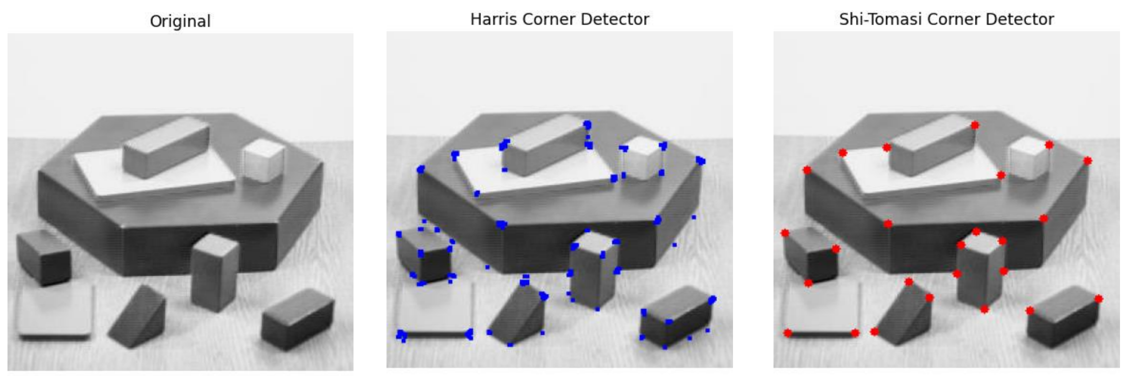

Figure 5: Corners detected by Harris Vs Shi-Tomasi Corner Detectors

3. Scale-Invariant Feature Transform (SIFT): While Harris and Shi-Tomasi detectors are rotation-invariant (because corners remain corners in rotated image also), they are sensitive to scaling.
   For example, in the example below you can see that a corner in a small image is flat when it is zoomed in the same window.

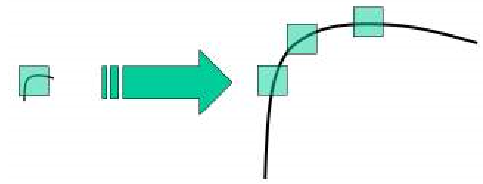

Figure 6: Impact of scaling on edges (Source: OpenCV SIFT Documentation)

SIFT can perform feature detection independent of viewpoint, depth, and scale of an image by transforming the image data into scale-invariant coordinates. Figure 7 shows the robustness of SIFT to scaling compared to Harris algorithm.

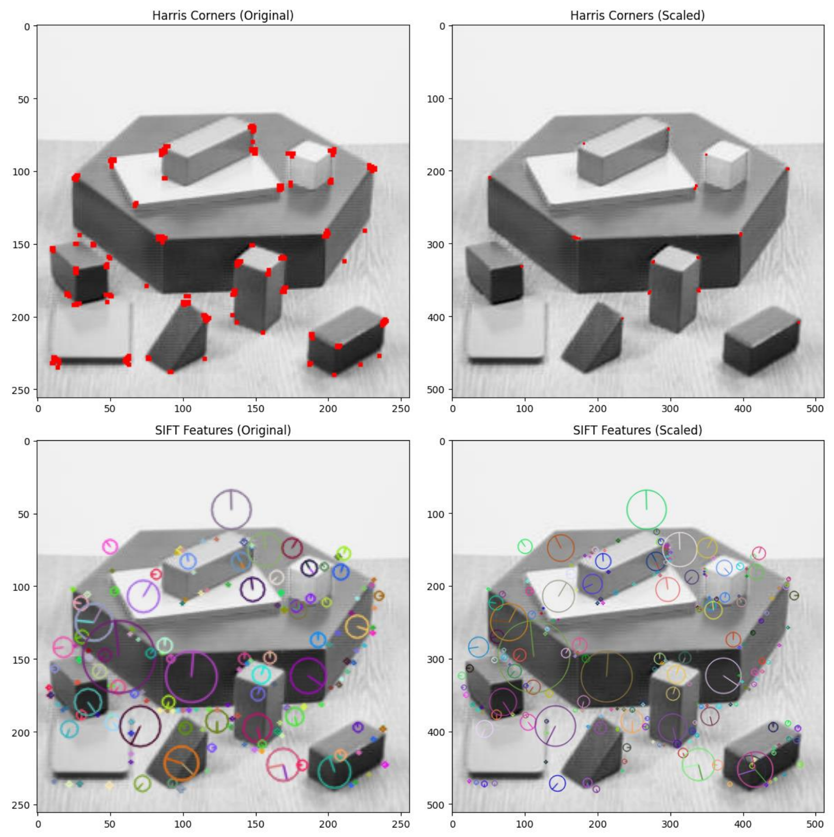

Figure 7: Corners detected by Harris Vs SIFT algorithms on different scaled images

Lab Task 1: Edge Detection

Use functions provided in OpenCV to complete the below tasks.

1.1 Identify the different edges present in an image using Sobel, Laplacian, and Canny edge detection algorithms, and discuss the differences in their outputs.

1.2 Using the provided image `jigsaw.jpg`, identify the boundary lines of the puzzle piece.
Follow the below steps to obtain the lines:

i. Crop the region containing the puzzle piece using a simple python matrix manipulation.

ii. Apply simple preprocessing techniques to convert the cropped image to grayscale and remove noises (e.g: Blurring, Thresholding).

iii. Perform edge detection on the binarized image using a suitable edge detection algorithm (e.g: Canny Edge Detection).

iv. Apply the Hough Transform (`cv2.HoughLines`) to detect lines that bound the four main corners of the jigsaw piece.
The final output should be similar to below:

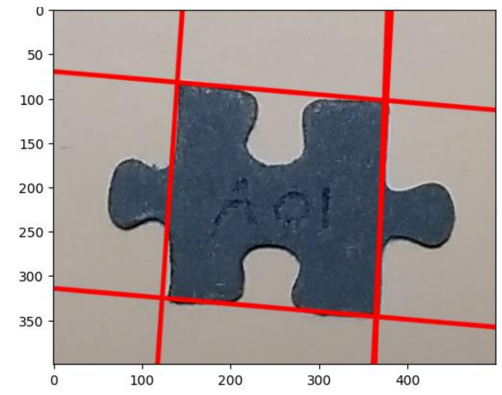

Figure 8: Example output of lab task 1.2

v. Explain the impact of the rho, theta, and threshold parameters of Hough transformation in detecting lines.

Lab Task 2: Corner Detection

Use functions provided in OpenCV to complete the below tasks.

2.1 Apply Harris, Shi-Tomasi, and SIFT algorithms on an image to identify corners and discuss the differences in these algorithms.

2.2 Using the provided image `jigsaw.jpg`, identify the corners present in the puzzle piece. The desired output is given in Figure 9:

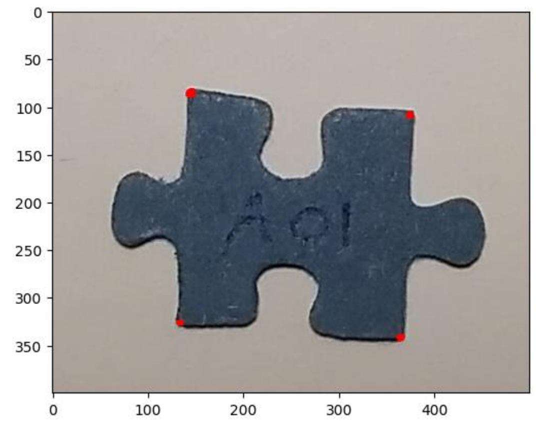

Figure 9: Example output of lab task 2.2

Submission

You need to submit all python files containing the relevant functions named according to the relevant question names or as indicated in the lab sheet, along with the main function to run them and display your outputs. Make sure to include the images you used to run the codes as well.

You need to submit a report (eYYXXXresults.pdf) including

- Screenshots for the codes
- All results from your code (your input and output images under each section after performing the required functions).
- Necessary explanations

under every part of the lab task.

You can submit a single ZIP file as eYYXXXlab06.zip including all:

- Python source codes
- All Input images
- Report

Note: YYXXX indicates your registration number in all cases

References:

1. https://docs.opencv.org/4.x/db/d27/tutorial_py_table_of_contents_feature2d.html
2. https://towardsdatascience.com/solving-jigsaw-puzzles-with-python-and-opencv-d775ba730660
3. https://github.com/cezannec/Computer-Vision-Exercises/tree/master