COMPARISON OF EDGE AND CORNER DETECTORS

NOVEMBER 12, 2021

Submitted to: Prof. Dr. Mustafa Ünel

Name of Student: Moses Chuka Ebere
Student Number: 31491
Course: Computer Vision (EE 417)

TABLE OF CONTENTS

INTRODUCTION

This is a computational assignment that focuses on the implementation of corner and edge detection algorithms.

Edge Detection

First derivative edge detection methods like Prewitt, Roberts, and Canny were implementation alongside the Laplacian of Gaussian edge detector, which is a second derivative edge detector.
For the above MATLAB's inbuilt edge function was used.

Corner Detection

Harris Corner Detection and Kanade-Tomasi Corner Detection were explored using different functions in MATLAB. First, the corner function (from the image processing toolbox) was implemented. Then, two functions – detectHarrisFeatures and detectMinEigenFeatures – from the Computer Vision Toolbox were applied.

The above were performed on several images, and the results were compared.

Finally, a succinct discussion on the results was included.

MATLAB CODES

For edge detection, I created two Matlab scripts – one function called Edge_Detection and a main script to call the function. Also, for corner detection, I created two Matlab scripts – one function called Corner_Detection and a main script to call the function.

The function in both cases accepts an image, smoothens it using Matlab's 'imgaussfilt' function, and converts it to grayscale, before detecting the edges/corners. After this, plots of the results (in different figures as instructed) are made within the function. Additionally, for the sake of comparison, subplots are made to place the detection results in the same figure.

In the two main scripts (called MainScript_Edge_Detection and MainScript_Corner_Detection), several images are read using the 'imread' function, and the functions are called on all the images.

Note: The reason I used two m-files for each case is to avoid excessive code repetition since the same task was to be performed on multiple images.

- *Just in case (to be on the safe side), I included additional (single) Matlab scripts – one for edge detection and one for corner detection - in the appendix that perform all the required tasks*.

Edge Detection

Here's the Edge_Detection function:

```
function Edge_Detection (img)

% Retain the Original Image for the Plot
I = img;

% First of all, smoothen the image using MATLAB's inbuilt
Gaussian filter.
```

```
% In the discussion, some results were obtained with
applying Gaussian
% filtering first for the sake of comparison.
I = imgaussfilt(I, 3);

% The row, column, and channels of the image are obtained
along with the
% cardinality of the image.
[r, c, ch] = size(I);
Card = r*c;

% This is added in case the image introduced is an RGB
image.
% It functions to convert it to a gray-scale image.
if (ch == 3)
    I = rgb2gray(I);
end

% Find the Edges using Prewitt, Roberts, Canny, and log
% First Derivative Edge Detectors
P = edge(I, 'Prewitt');
R = edge(I, 'Roberts');
C = edge(I, 'Canny');
% Second Derivative Edge Detector
L = edge(I, 'log');

% Plots - Each image will be in a different figure
figure %1 - Original Image
imshow(img)
title('Original Image')

figure %2
imshow(P)
title('Prewitt Edges')

figure %3
imshow(R)
title('Roberts Edges')

figure %4
imshow(C)
title('Canny Edges')
```

```
figure %5
imshow(L)
title('Laplacian of Gaussian Edges')

% Subplots are used to place the above results on the same
figure for the
% sake of comparison.
figure %5
subplot(2,2,1)
imshow(P)
title('Prewitt Edges')

subplot(2,2,2)
imshow(R)
title('Roberts Edges')

subplot(2,2,3)
imshow(C)
title('Canny Edges')

subplot(2,2,4)
imshow(L)
title('Laplacian of Gaussian Edges')

end
```

Here's the main script that reads in the images and calls the function.

```
% Edge Detection

clear all
clc

% Read the images whose edges will be detected
a = imread('bridge.jpg');
a1 = imread('building.jpg');
a2 = imread('beach.jpg');
a3 = imread('library.jpg');
a4 = imread('lego1.jfif');
a5 = imread('steps.jpg');
```

```
% Call the Edge Detection Function
Edge_Detection(a)
Edge_Detection(a1)
Edge_Detection(a2)
Edge_Detection(a3)
Edge_Detection(a4)
Edge_Detection(a5)
```

Corner Detection

Here's the Corner_Detection function:

```
function Corner_Detection (img)

% Retain the Original Image for the Plot
I = img;

% First of all, smoothen the image using MATLAB's inbuilt
Gaussian filter.
% In the discussion, some results were obtained with
applying Gaussian
% filtering first for the sake of comparison.
I = imgaussfilt(I, 3);

% The row, column, and channels of the image are obtained
along with the
% cardinality of the image.
[r, c, ch] = size(I);
Card = r*c;

% This is added in case the image introduced is an RGB
image.
% It functions to convert it to a gray-scale image.
if (ch == 3)
    I = rgb2gray(I);
end

% Detect the Corners
% Using Corner function
C_H = corner(I, 'Harris');
C_M = corner(I, 'MinimumEigenvalue');
```

```
% Using Functions from the Computer Vision Toolbox
H = detectHarrisFeatures(I);
ME = detectMinEigenFeatures(I);

% Plots - Each image will be in a different figure
figure %1 - Original Image
imshow(img)
title('Original Image')

figure %2
imshow(img)
hold on
plot(C_H(:,1),C_H(:,2),'r*');
title('Corner Detection Using Corner Function')
xlabel('Method - Harris')

figure %3
imshow(img)
hold on
plot(C_M(:,1),C_M(:,2),'r*');
title('Corner Detection Using Corner Function');
xlabel('Method - MinimumEigenvalues')

figure %4
imshow(img)
hold on;
plot(H.selectStrongest(100));
title('Harris Corner Detection Using
detectHarrisFeatures')

figure %5
imshow(img)
hold on;
plot(ME.selectStrongest(100));
title('Kanade-Tomasi Corner Detection Using
MinEigenFeatures')

% Subplots are used to place the above results on the same
figure for the
% sake of comparison.
figure %6
subplot(2,2,1)
```

```
imshow(img)
hold on
plot(C_H(:,1),C_H(:,2),'r*');
title('Corner Detection Using Corner Function (Method -
Harris)')

subplot(2,2,2)
imshow(img)
hold on
plot(C_M(:,1),C_M(:,2),'r*');
title('Corner Detection Using Corner Function (Method -
MinimumEigenvalues)');

subplot(2,2,3)
imshow(img)
hold on;
plot(H.selectStrongest(100));
title('Harris Corner Detection Using
detectHarrisFeatures')

subplot(2,2,4)
imshow(img)
hold on;
plot(ME.selectStrongest(100));
title('Kanade-Tomasi Corner Detection Using
MinEigenFeatures')

end
```

Here's the main script that reads in the images and calls the function.

```
% Corner Detection

clear all
clc

% Read the images whose edges will be detected
a = imread('bridge.jpg');
a1 = imread('building.jpg');
a2 = imread('beach.jpg');
a3 = imread('library.jpg');
```

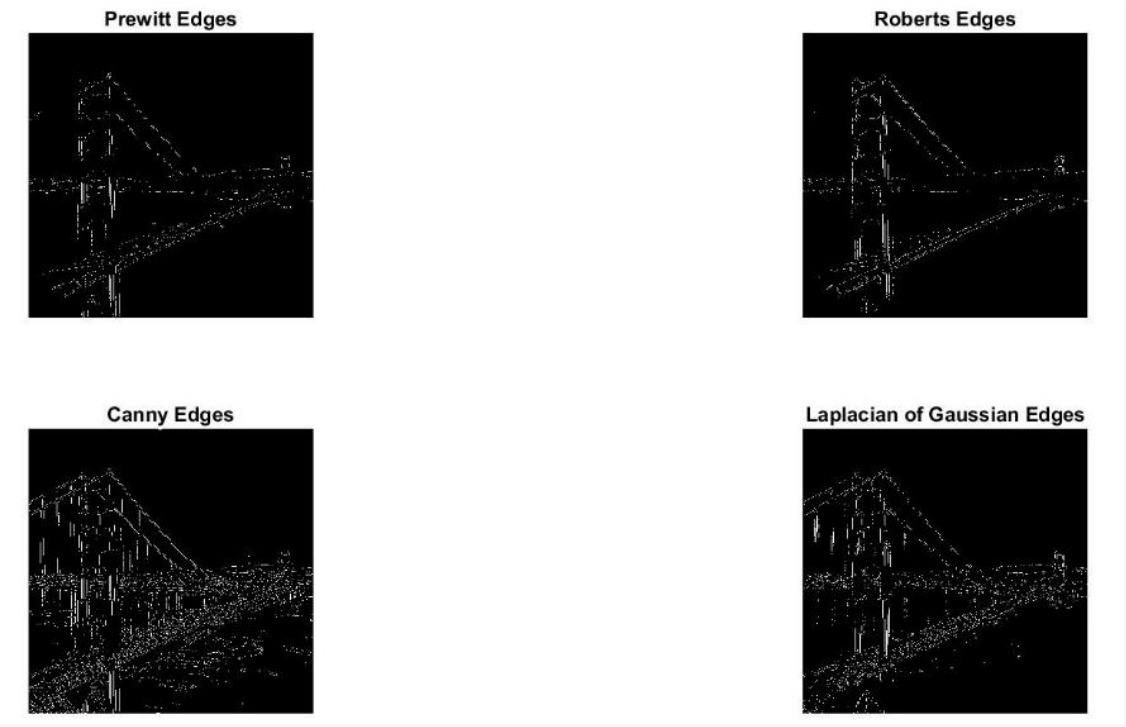
```
a4 = imread('lego1.jfif');
a5 = imread('steps.jpg');

% Call the Edge Detection Function
Corner_Detection(a)
Corner_Detection(a1)
Corner_Detection(a2)
Corner_Detection(a3)
Corner_Detection(a4)
Corner_Detection(a5)
```

RESULTS

Edge Detection

Image 1: The Golden State Bridge



Comments and Comparison

As seen above, all the detectors applied were able to detect some edges in the bridge image; however, we notice significant differences in the performance of each algorithm.

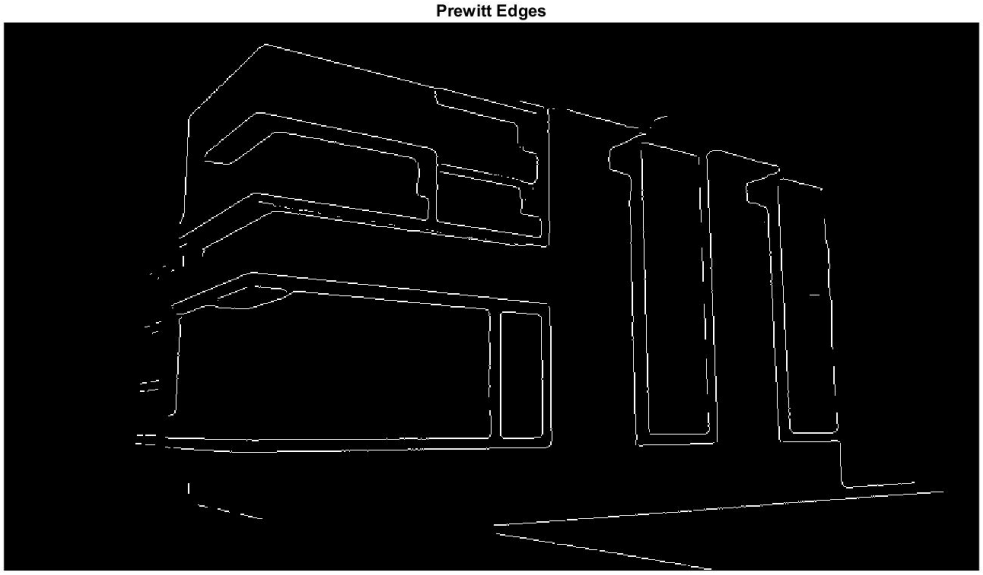
We can clearly observe that the Canny Edge detector detects the most edges, followed by the Laplacian of Gaussian Detector. The Prewitt and Roberts algorithms come in last, with the Prewitt filter detecting slightly more edges than the Roberts filter. This is easily noticeable at the top-left corner of the results.

Although the Canny algorithm detects the most edges, it is difficult to decipher the exact object whose edges were detected in some regions of the result (like in the horizon and city skyline in the background). This comment is also applicable to the LoG-filtered image.

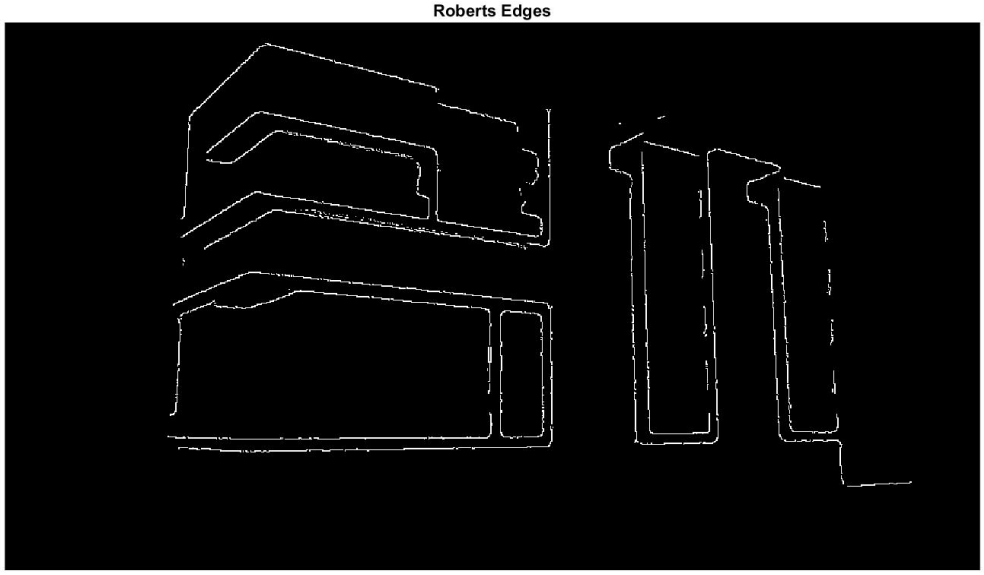
Image 2: A building

Original Image

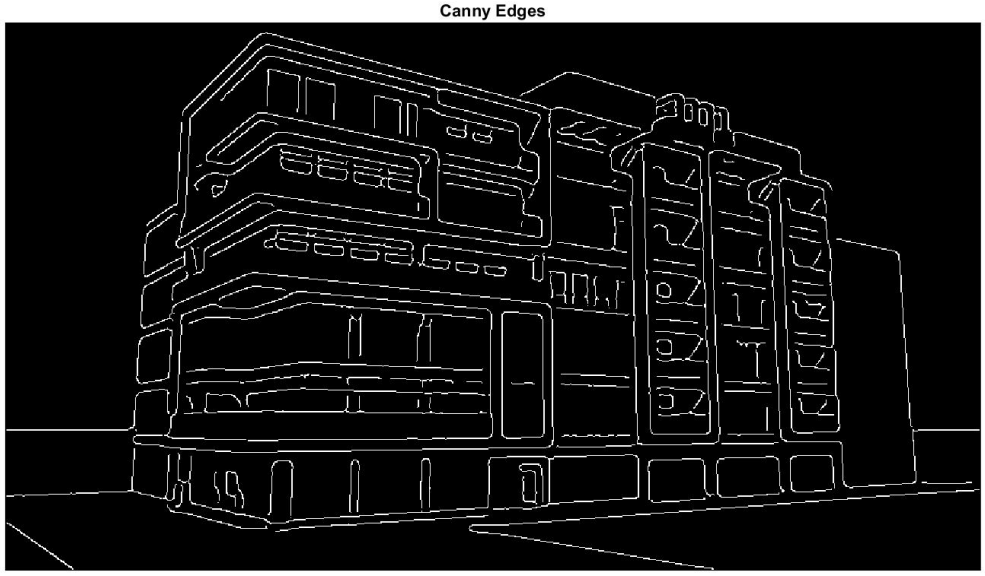
Prewitt Edges



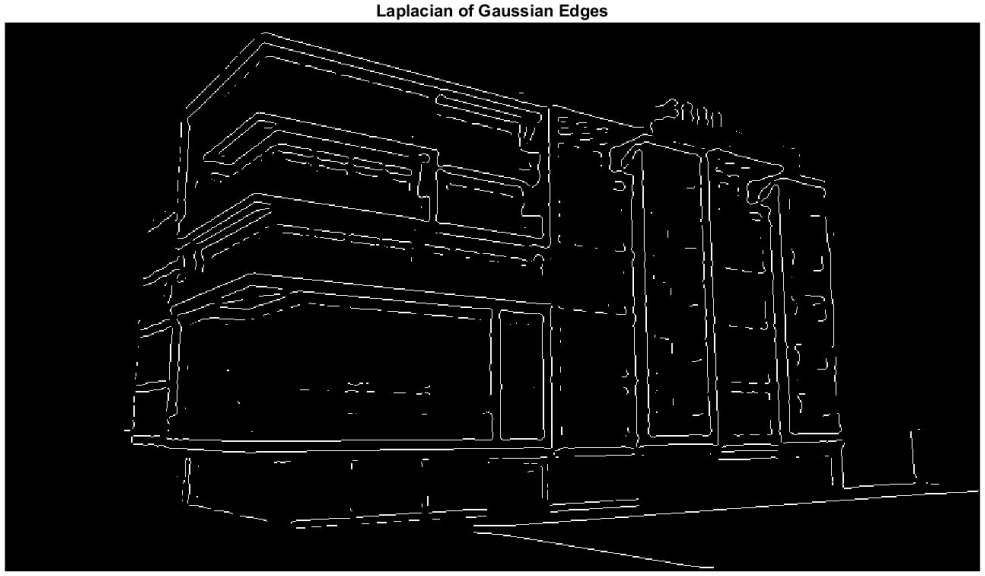
Roberts Edges



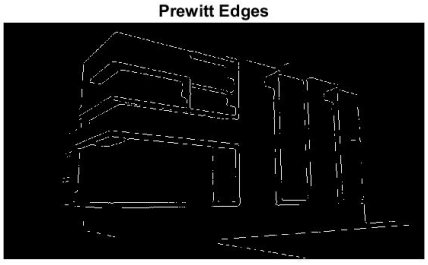
Canny Edges



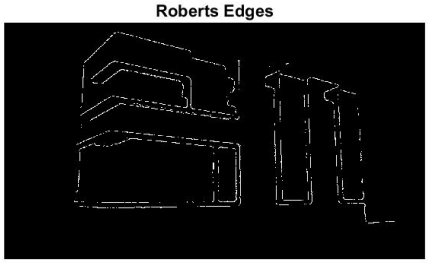
Laplacian of Gaussian Edges



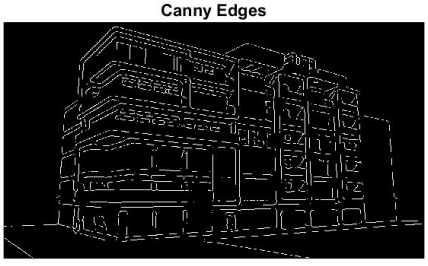
Prewitt Edges



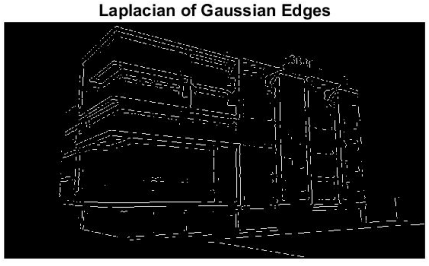
Roberts Edges



Canny Edges



Laplacian of Gaussian Edges



Comments and Comparison

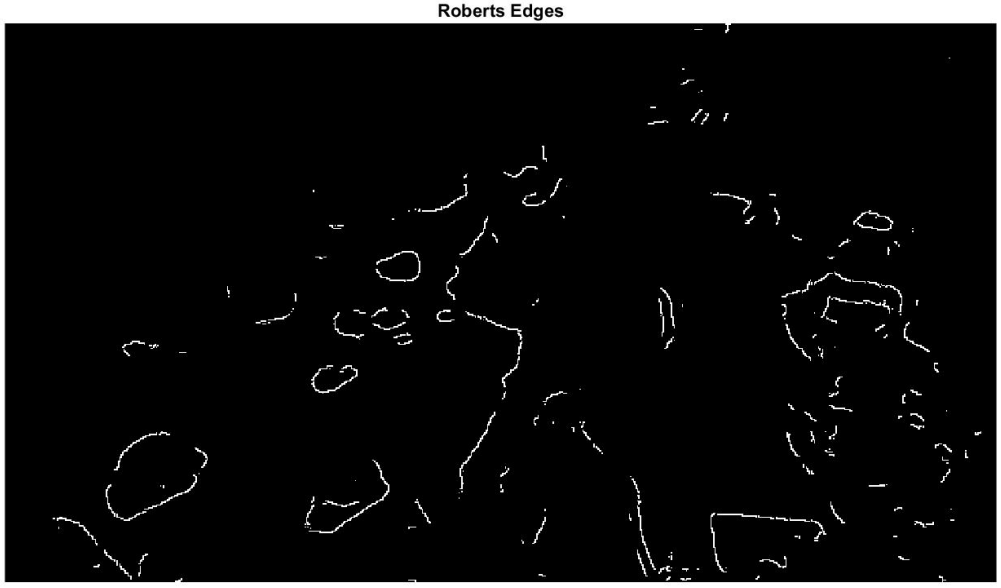
As seen above, all the detectors applied were able to detect some edges in the building; however, we notice significant differences in the performance of each algorithm.

We can clearly observe that the Canny Edge detector detects the most edges, followed by the Laplacian of Gaussian Detector. The Prewitt and Roberts algorithms come in last, with the Prewitt filter detecting slightly more edges than the Roberts filter. This is easily noticeable around some subtle features of the building like the glass windows.

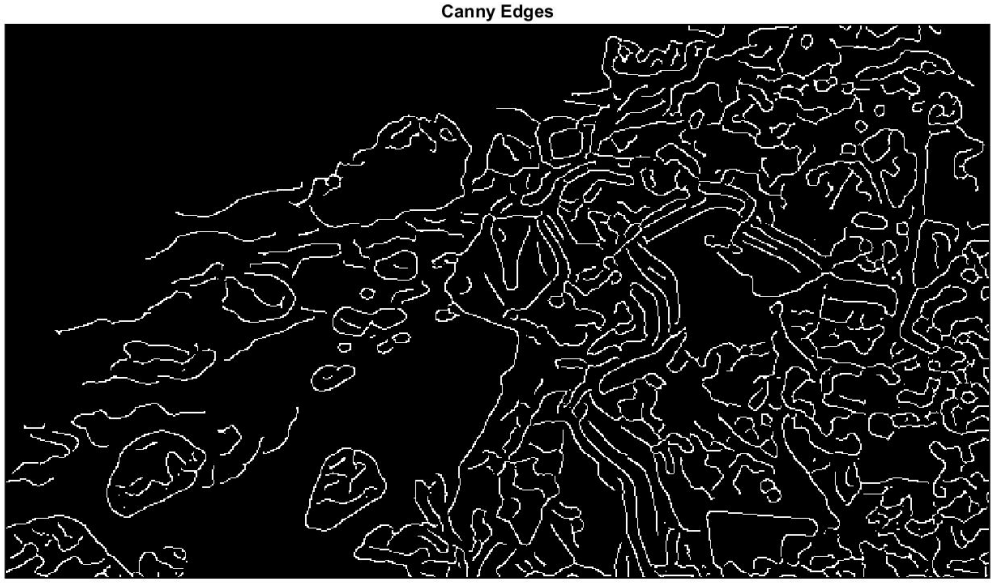
With the Canny algorithm, some parts of the landscape around the buildling are clearly noticeable. One who's not familiar with the silhouette of a building might not be able to conclude that the results from the Roberts and Prewitt filters are full buildings.

Image 3: A beach

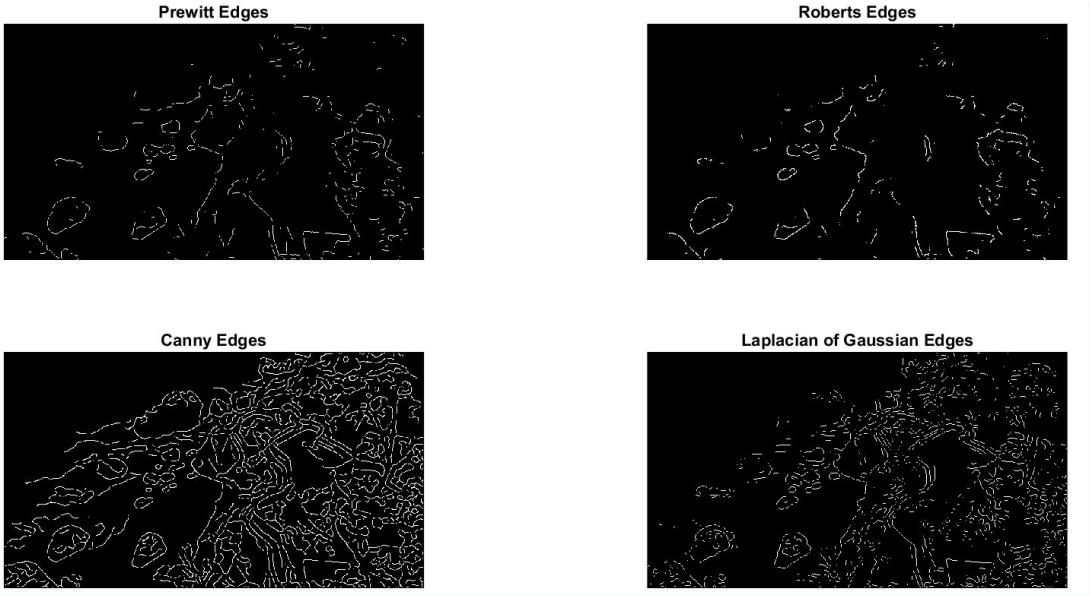
Roberts Edges



Canny Edges



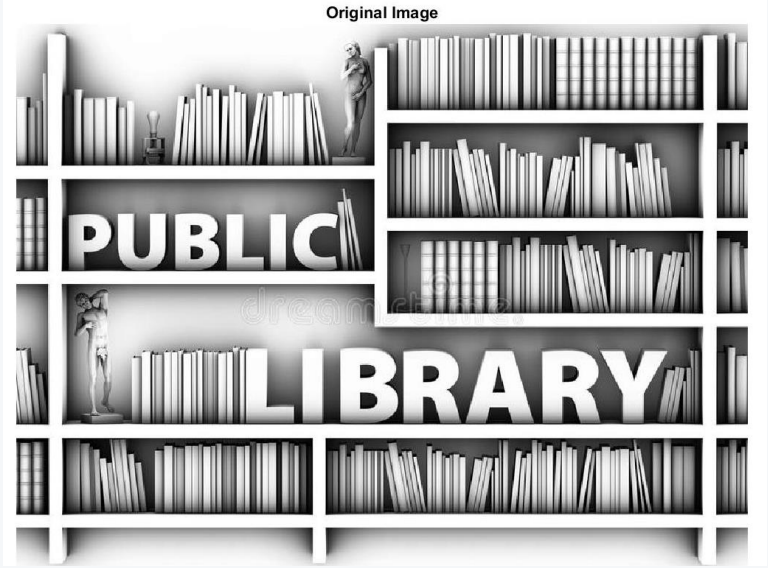
Laplacian of Gaussian Edges



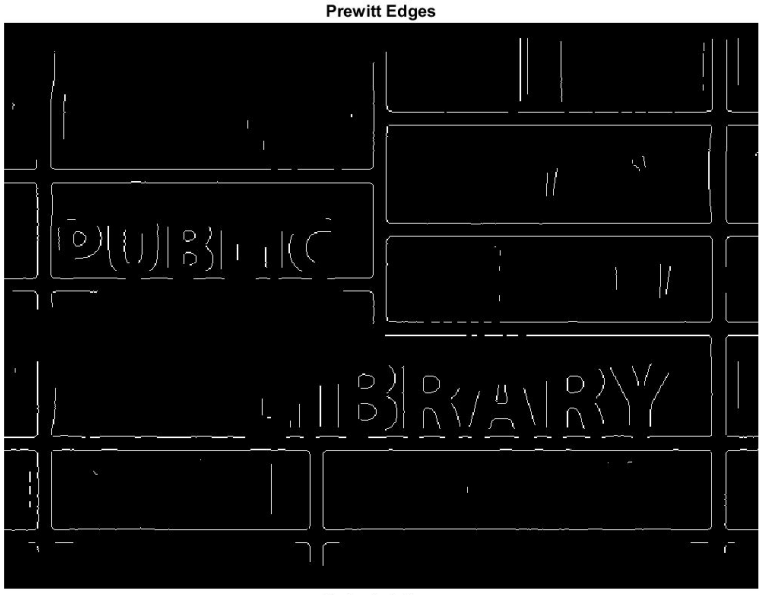
This photo may not be so suitable for edge detection given the similarity in the contours of different objects in the image. Just looking at the results, it will be quite impossible to tell that the original image was that of a beach.

In terms of performance of the detector, Canny is ahead of the other three as it picks up several edges. On the other end of the performance spectrum is the Roberts edge detector.

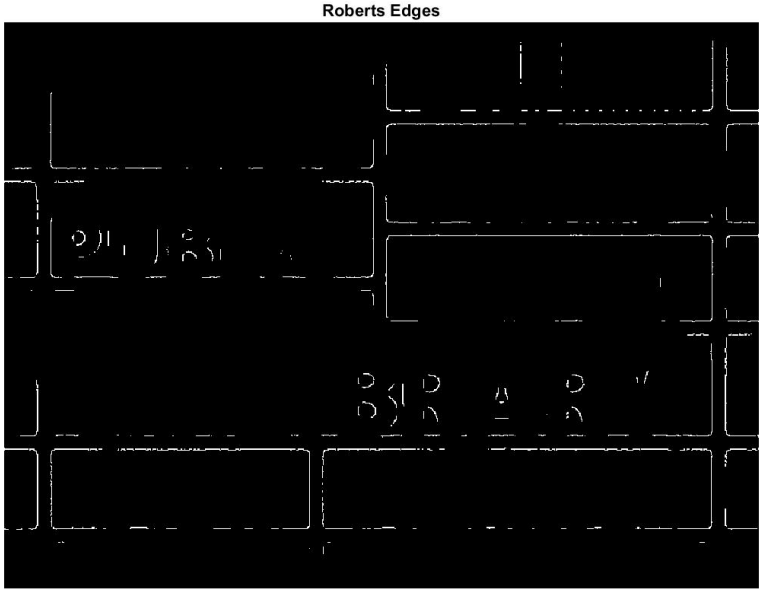
Image 4: A library



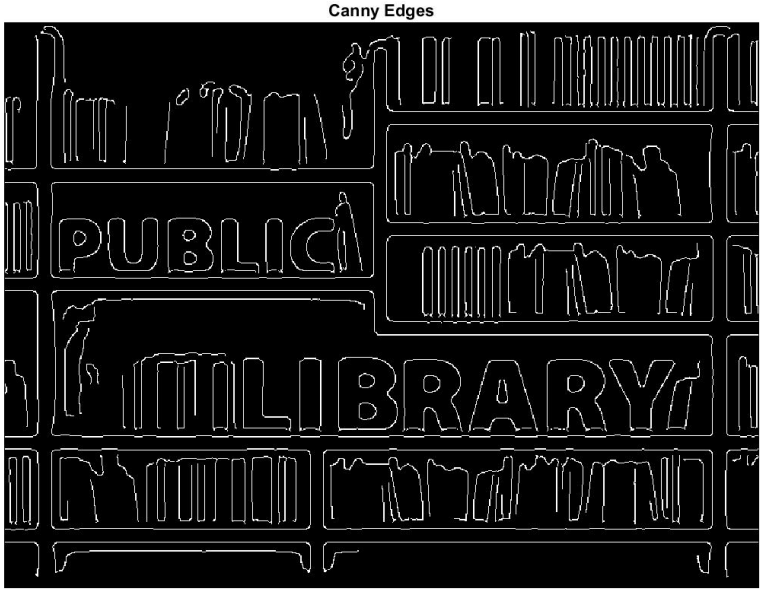
Prewitt Edges

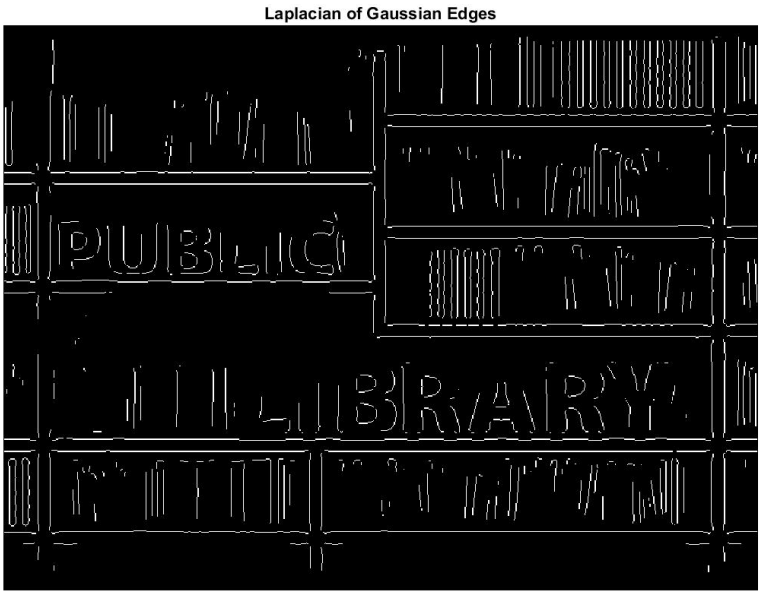


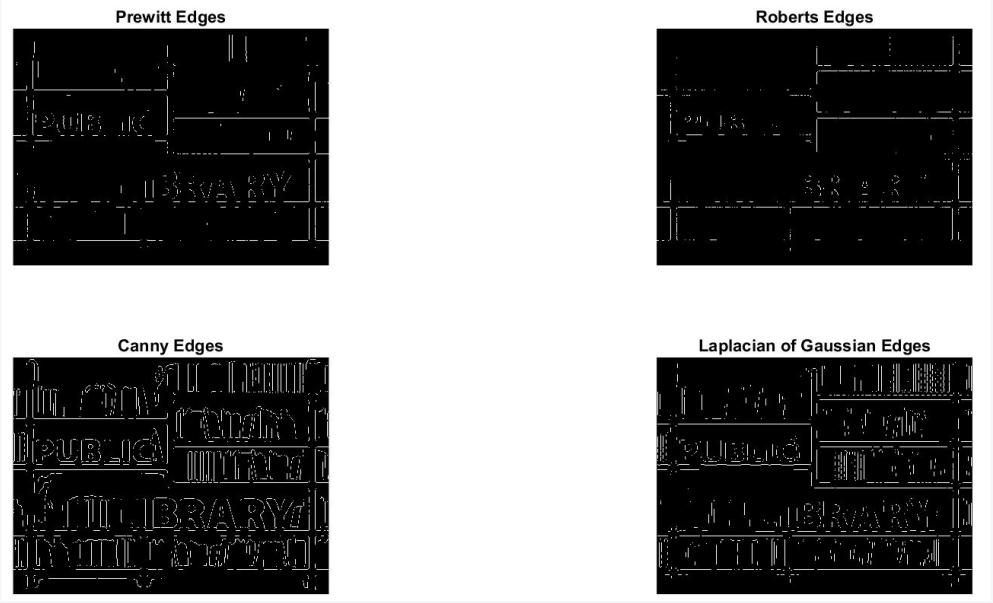
Roberts Edges



Canny Edges



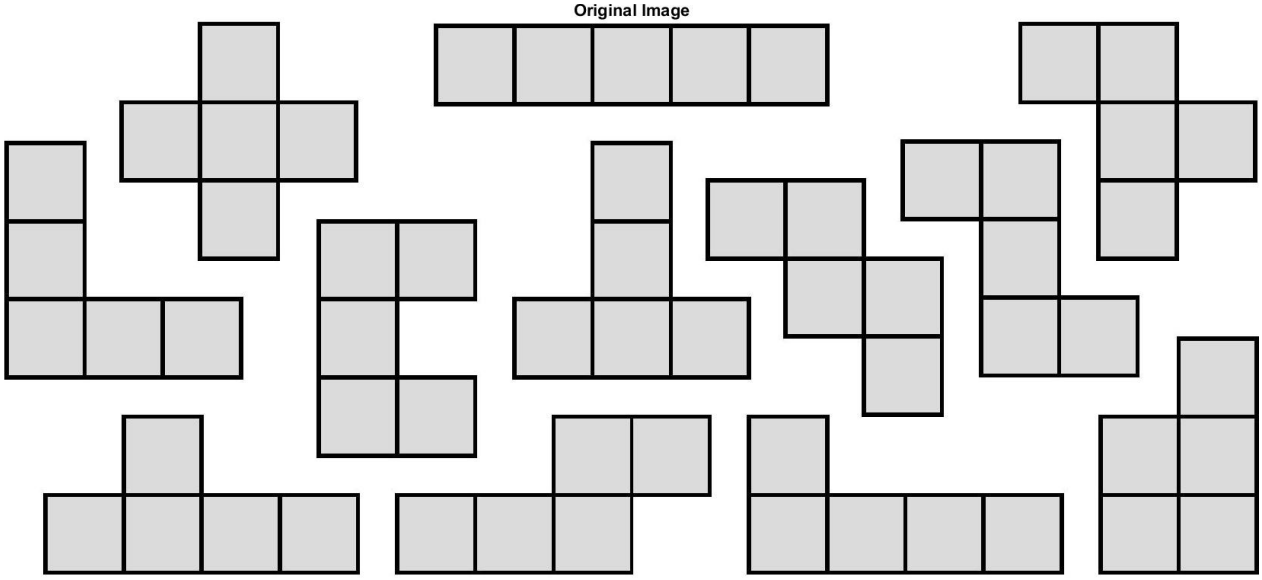




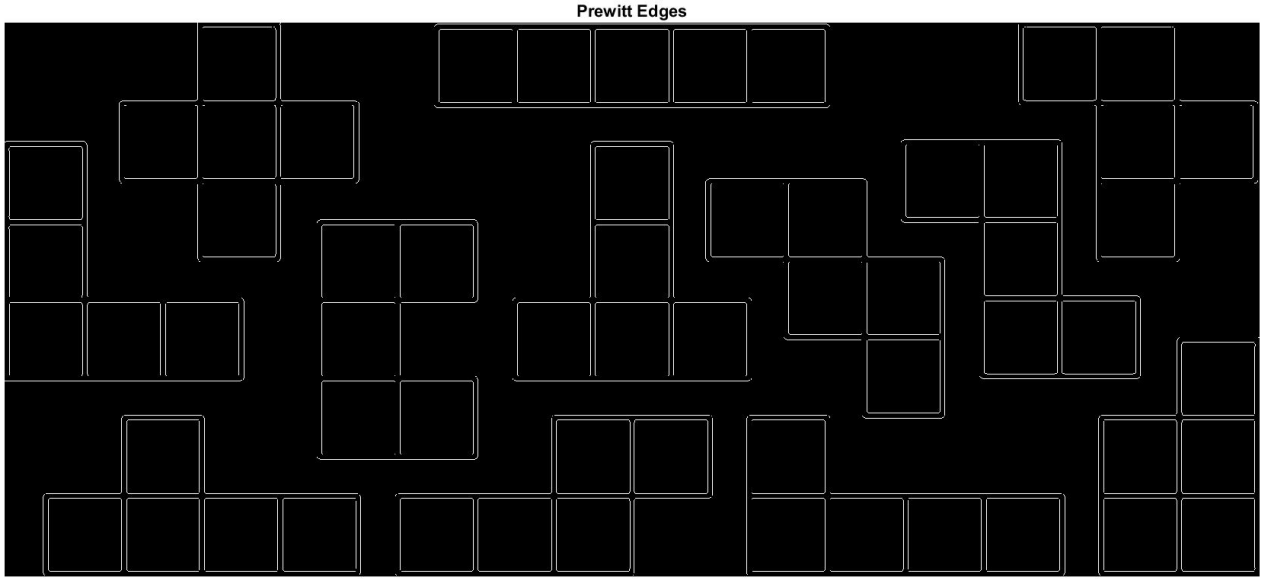
Similar comments as before can be made for this image. Notice how the Canny detector fully identifies the 'PUBLIC LIBRARY'.
One thing worthy of note is that in areas of sharp variations in pixel intensities, the detectors perform a lot better.

Image 5: Some lego blocks

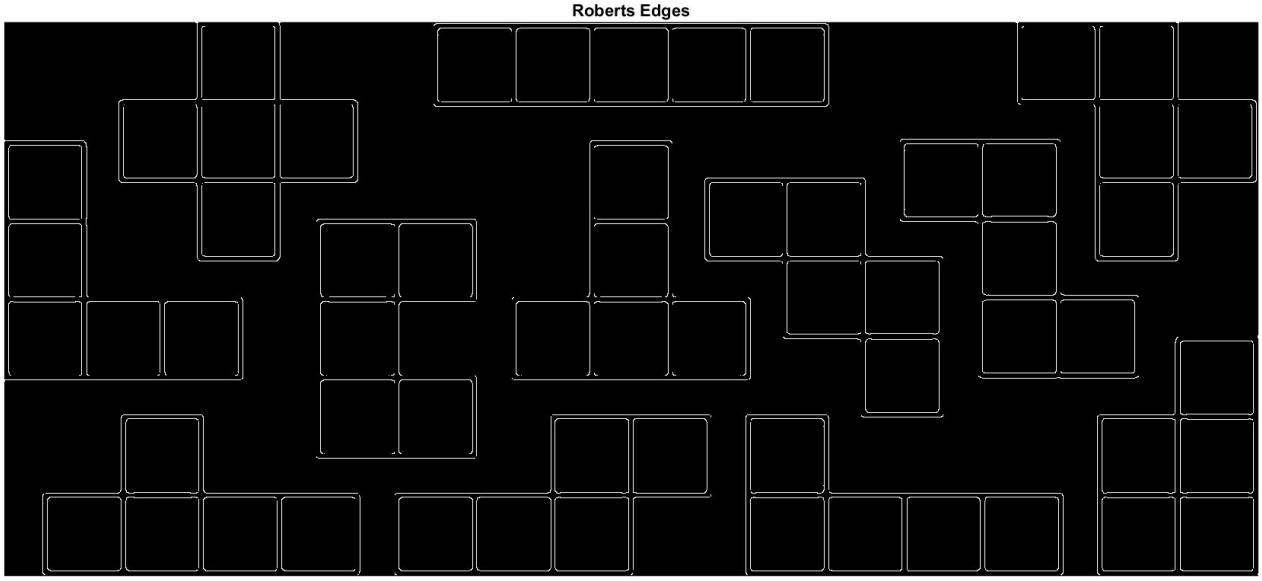
Original Image

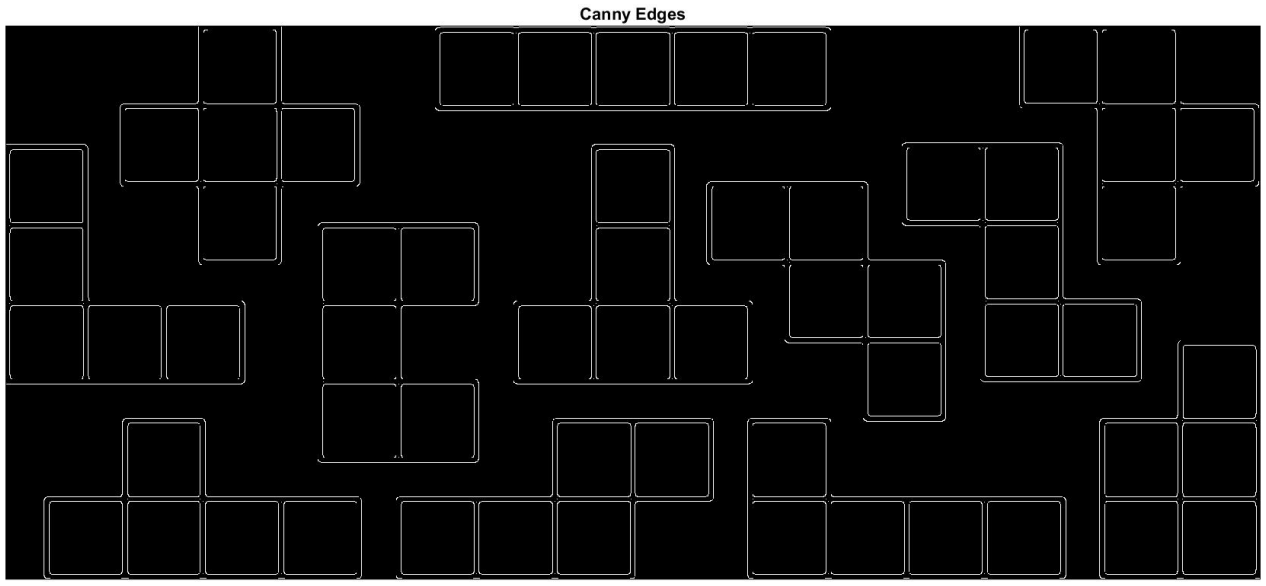


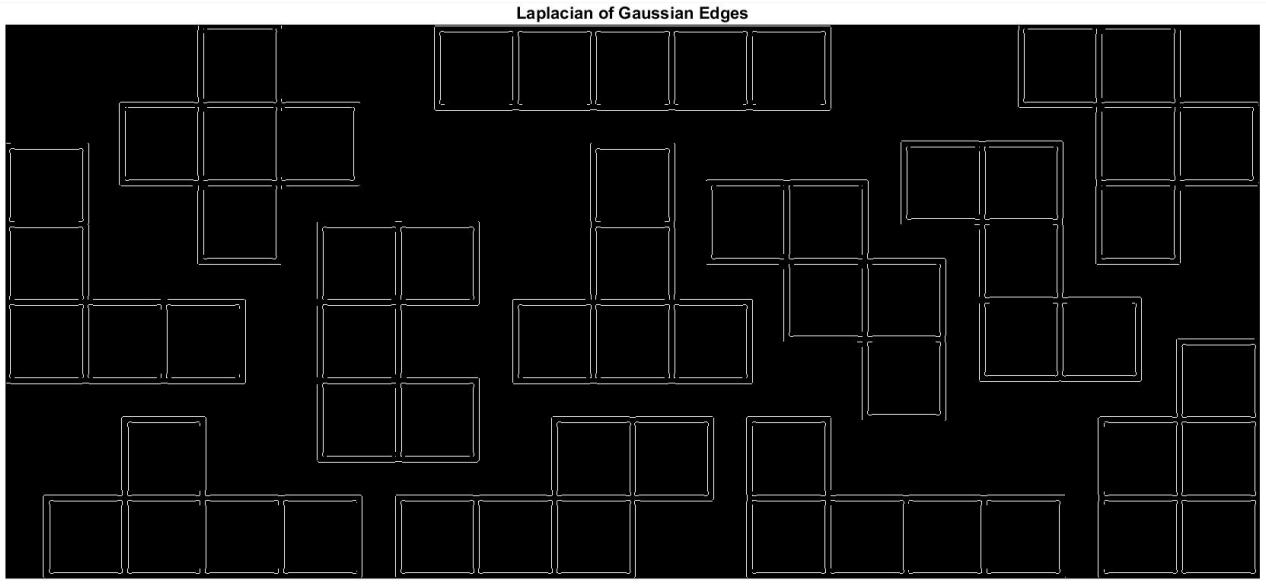
Prewitt Edges

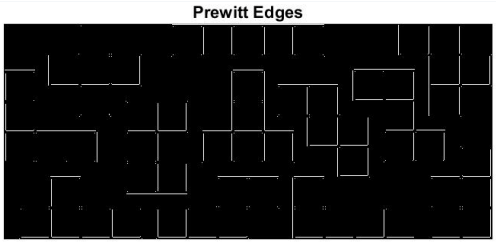


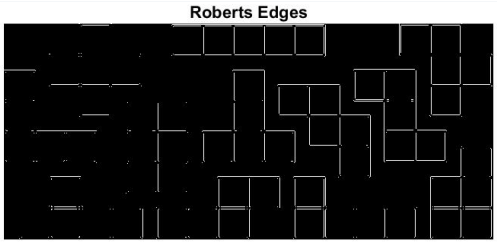
Roberts Edges

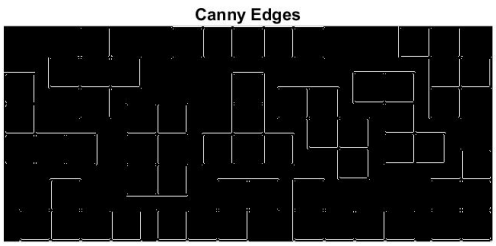


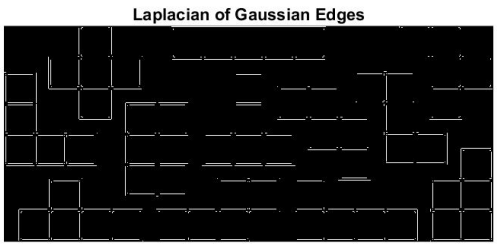








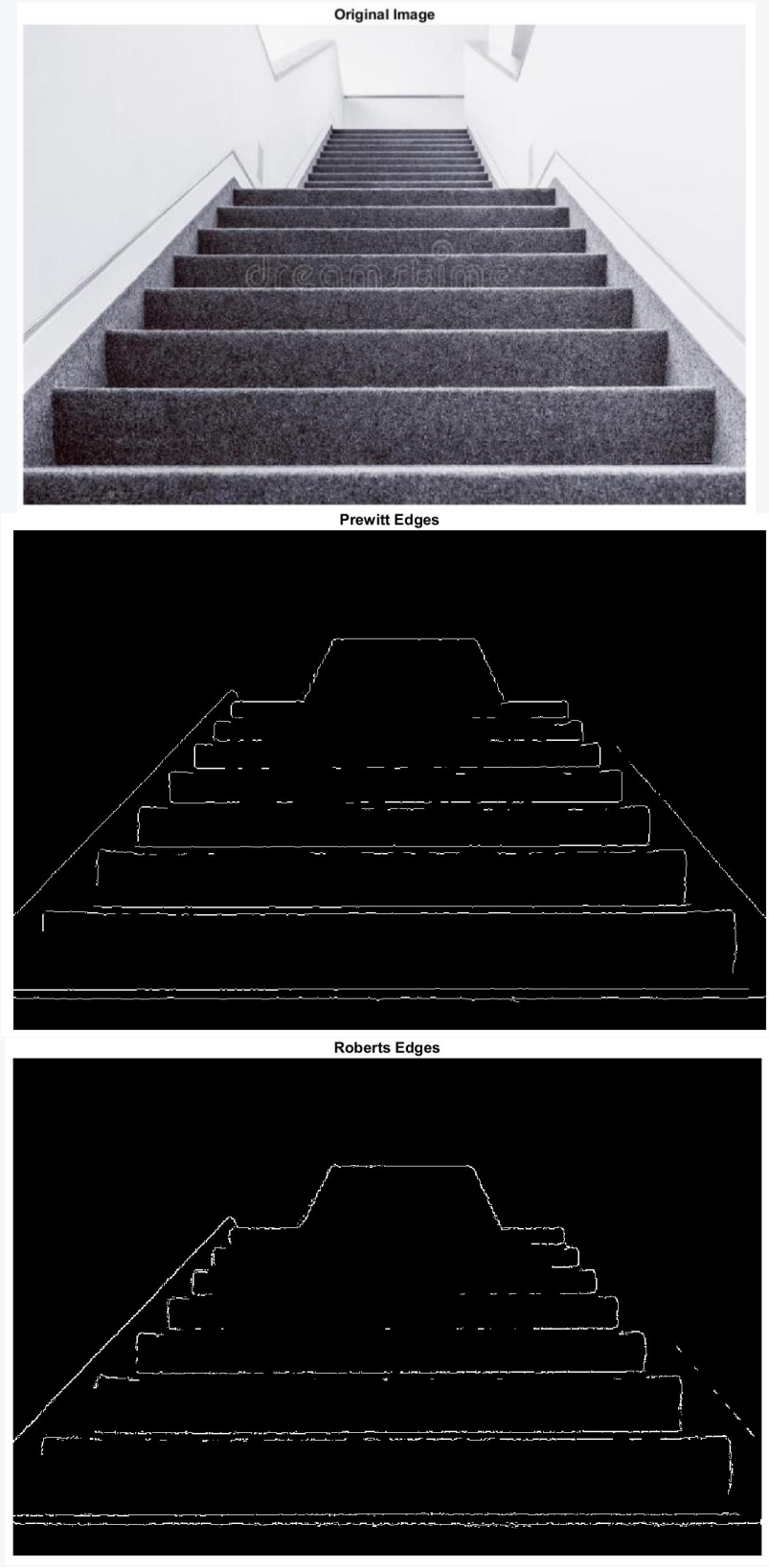


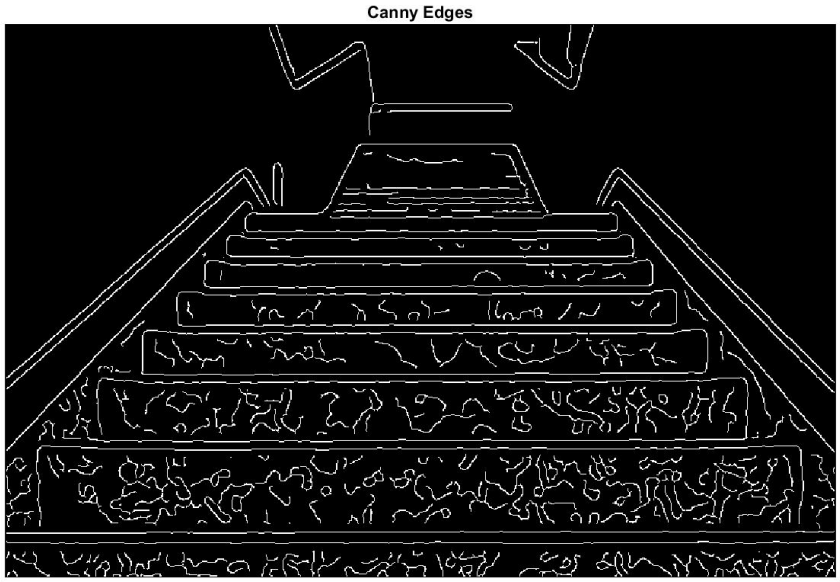


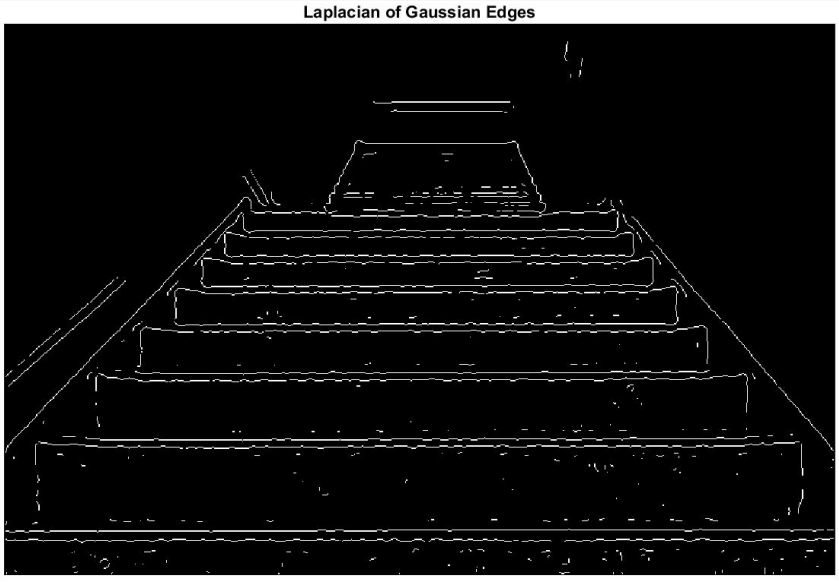
In this case, all detectors perform really well because of the clear outlines and contrast in the image. **We can opine that the performance**

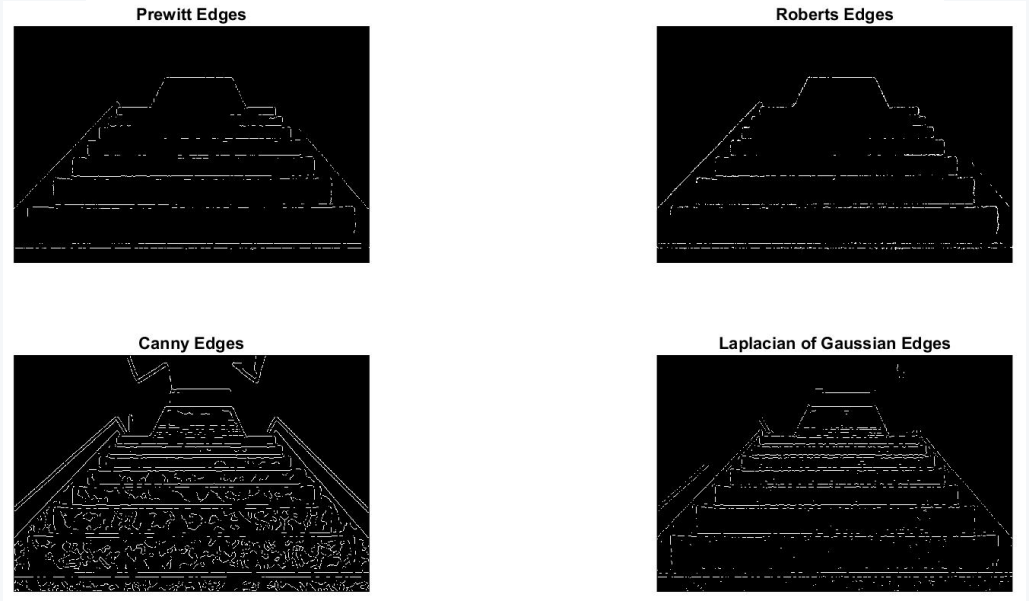
of an edge detector algorithm is relatively dependent on the image it is implemented on.

Image 6: A staircase







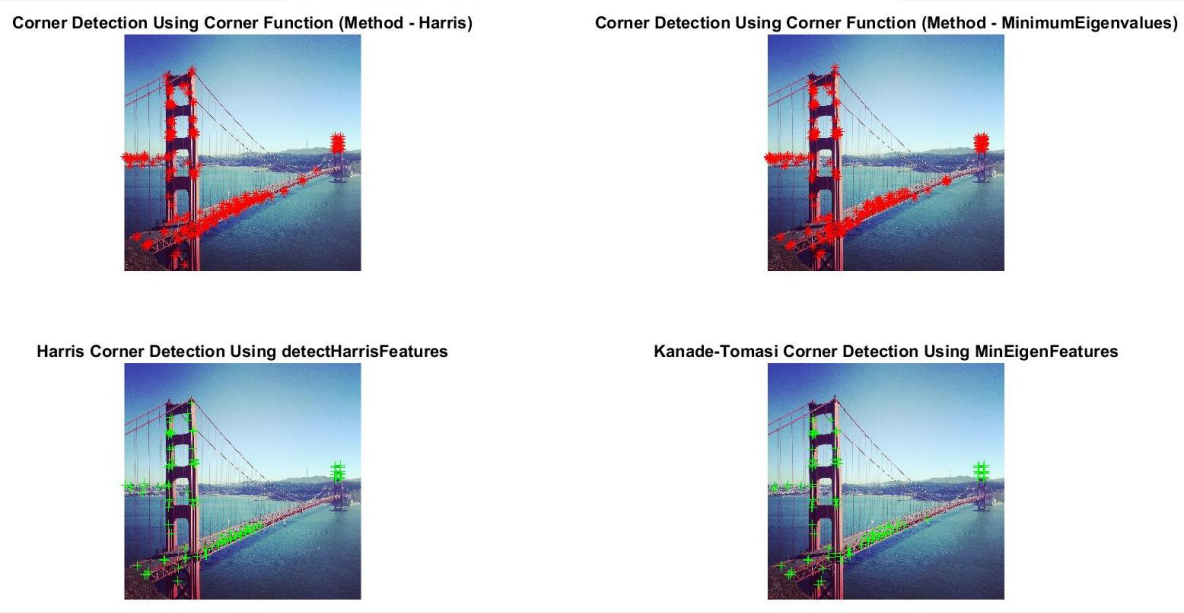


In this image, we notice a slight disadvantage of more precise edge detection algorithms. Notice how Canny detector highlights portions of the rug. In reality,

these are not edges. In my eyes, the Canny detector does an excessive job in the rug areas.

Corner Detection

Image 1: The Golden State Bridge



Normally, MATLAB doesn't recommmend the usage of the corner function.

In the above results, it is quite difficult to appreciate the performance of the detectors, but we notice that the corner function is prone to considering more points as corners compared to the detectHarrisFeatures and MinEigenFeatures functions.

Between the Harris detector and the Kanade-Tomasi Detector, we can observe that the Kanade-Tomasi algorithm is more precise. This is because the Harris operator computes an approximation of the minimum eigenvalues of the corner matrix.

Image 2: A building

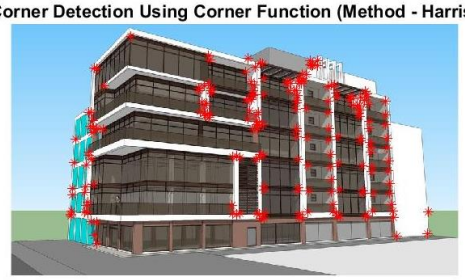
Method - Harris

Method - MinimumEigenvalues

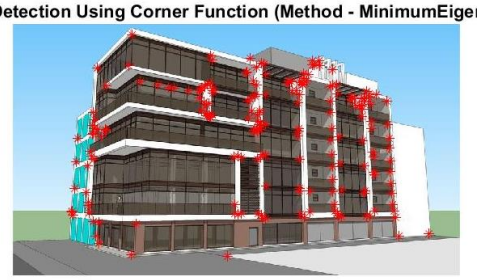
Harris Corner Detection Using detectHarrisFeatures

Kanade-Tomasi Corner Detection Using MinEigenFeatures

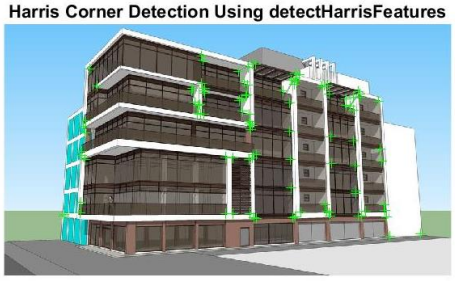
Corner Detection Using Corner Function (Method - Harris)



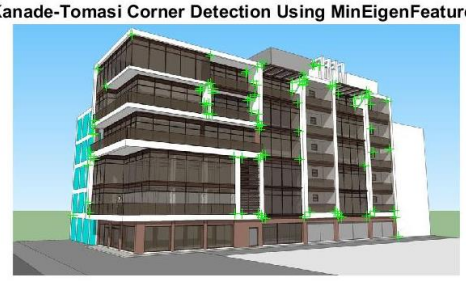
Corner Detection Using Corner Function (Method - MinimumEigenvalues)



Harris Corner Detection Using detectHarrisFeatures



Kanade-Tomasi Corner Detection Using MinEigenFeatures

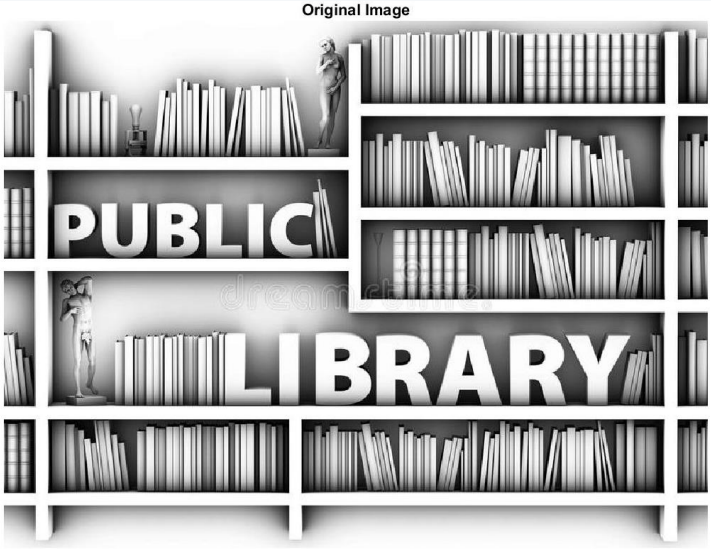


This is a really good image to test the performance of corner detectors. The MinEigenFeatures is more precise at detecting corners than the detectHarrisFeatures.

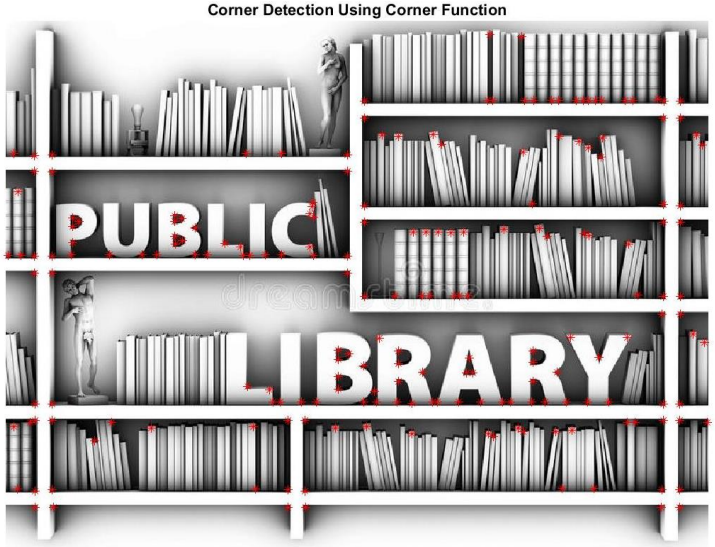
Image 3: A beach

Image 4: A library

Original Image

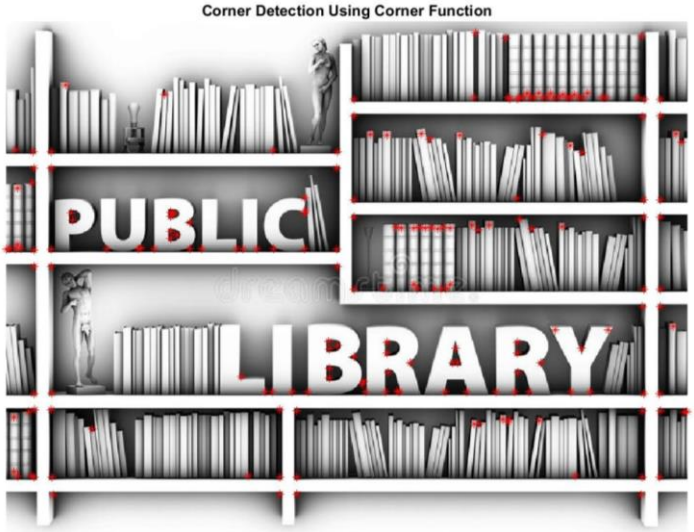


Corner Detection Using Corner Function



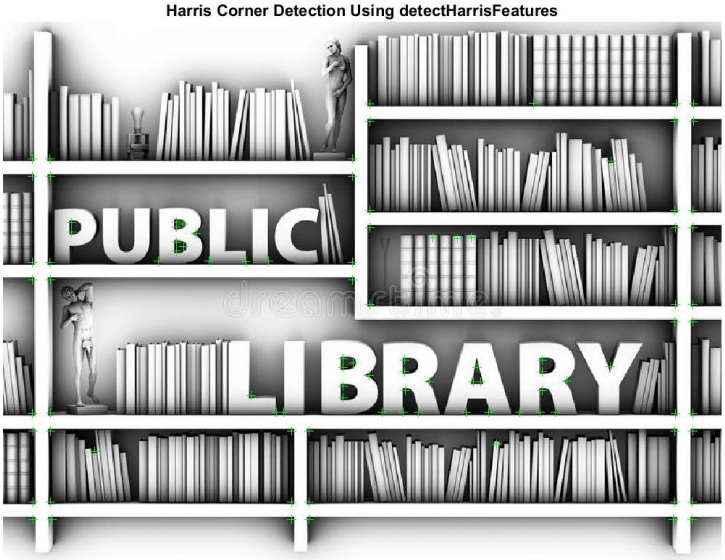
Method - Harris

Corner Detection Using Corner Function

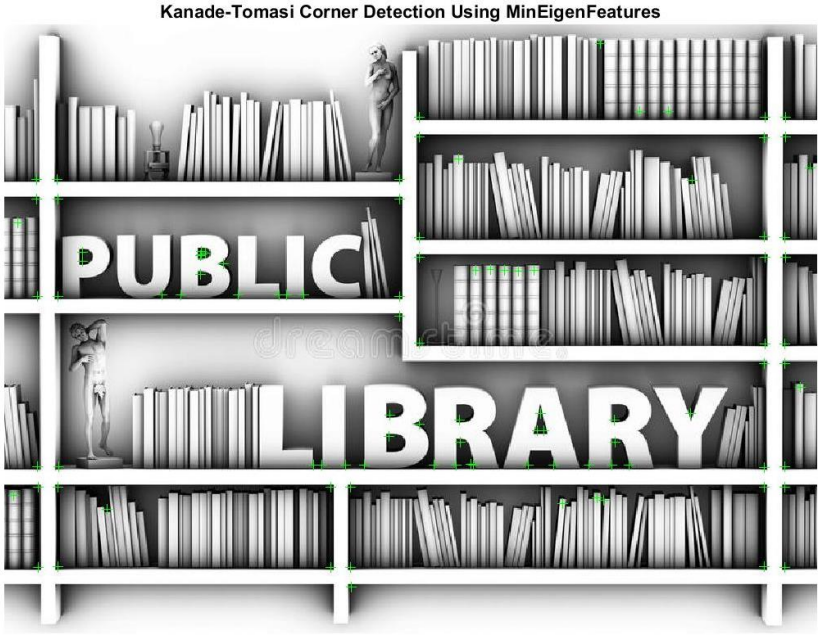


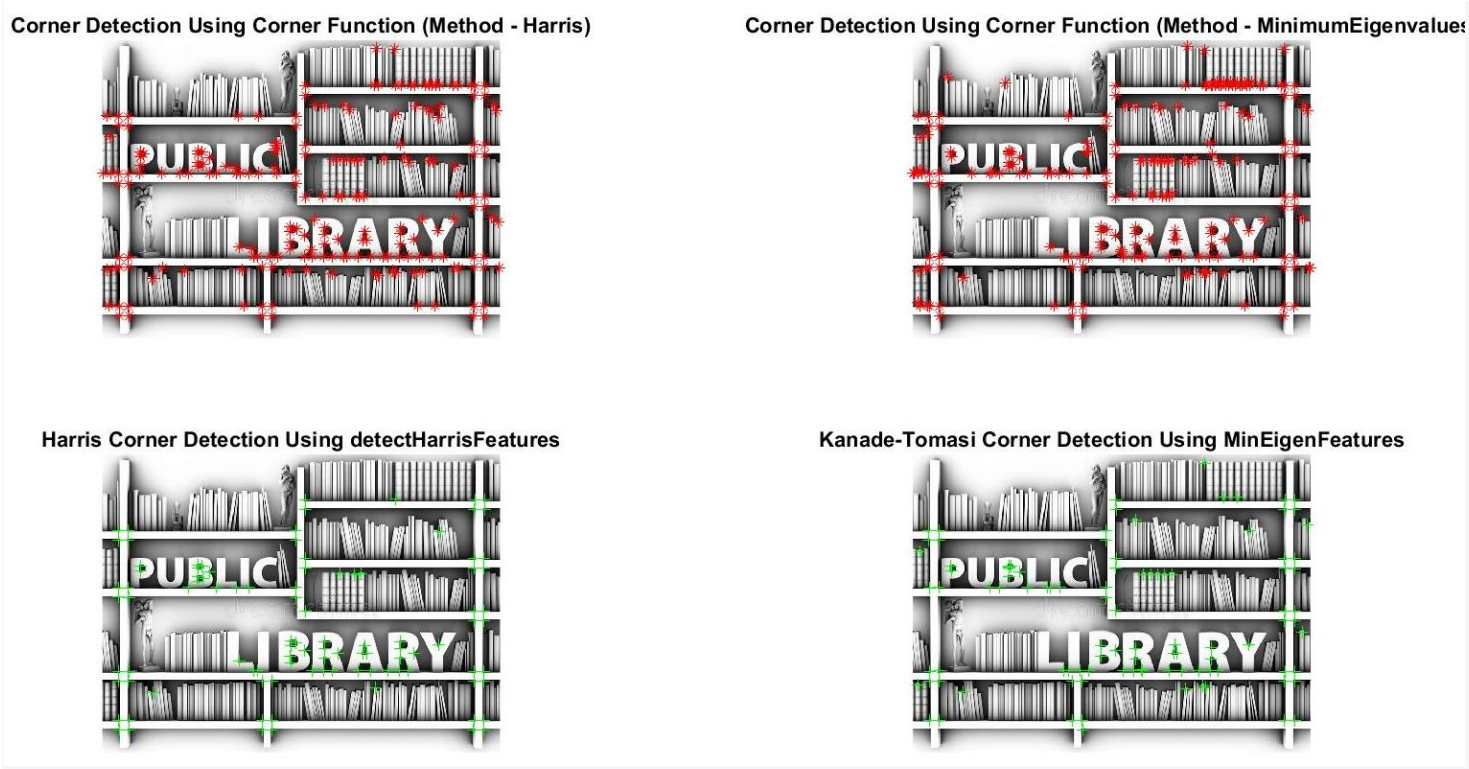
Method - MinimumEigenvalues

Harris Corner Detection Using detectHarrisFeatures



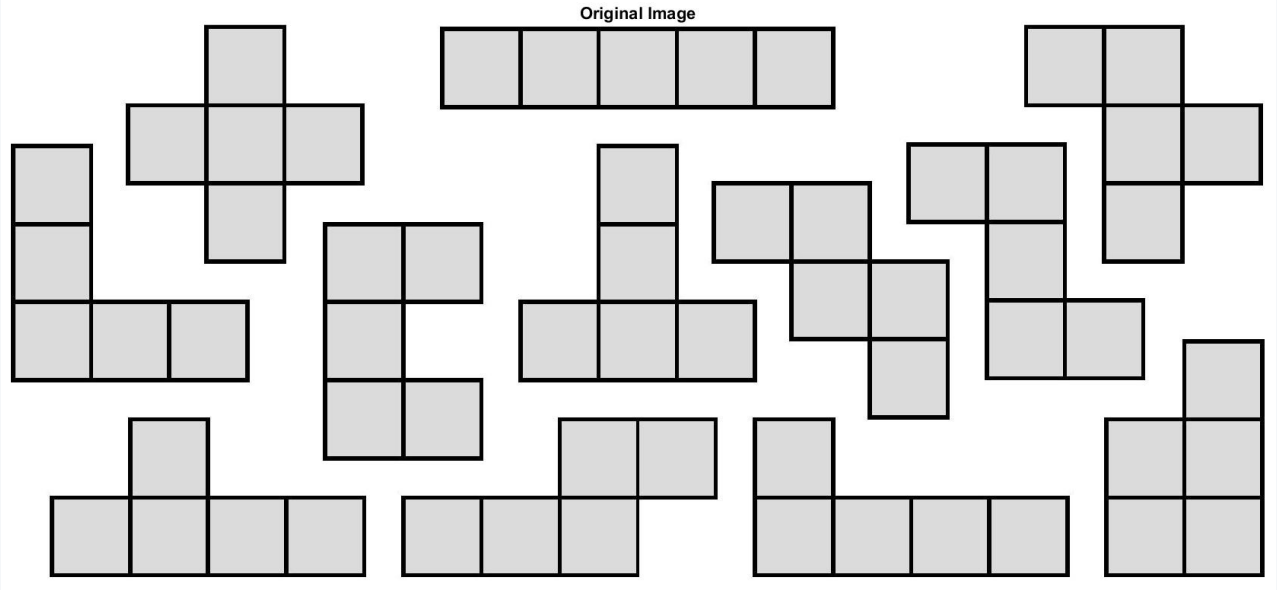
Kanade-Tomasi Corner Detection Using MinEigenFeatures

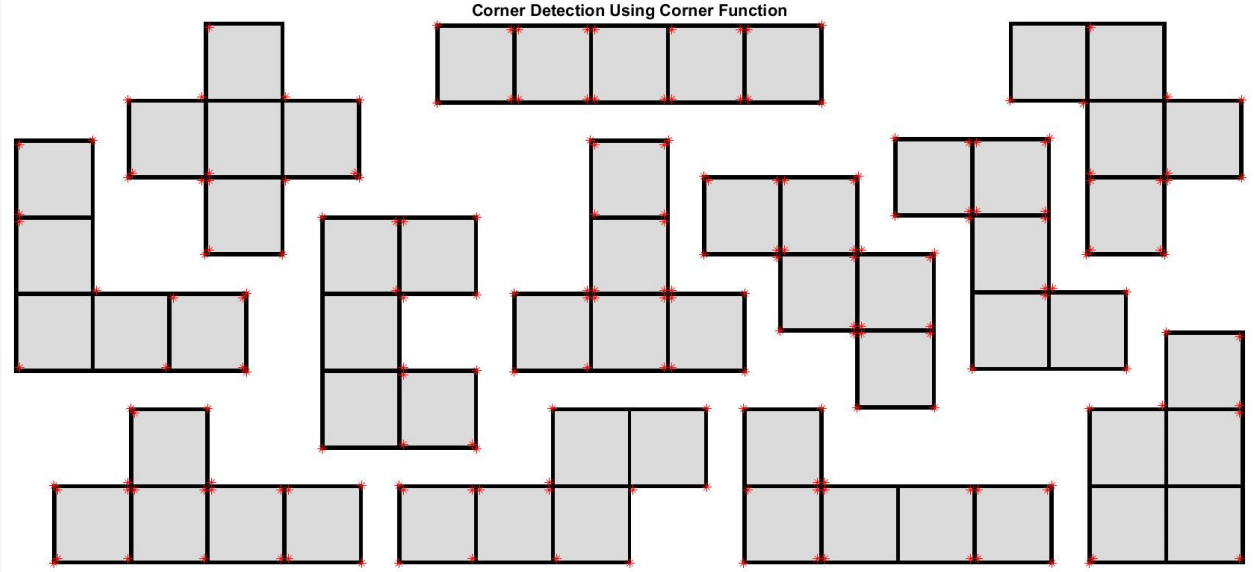




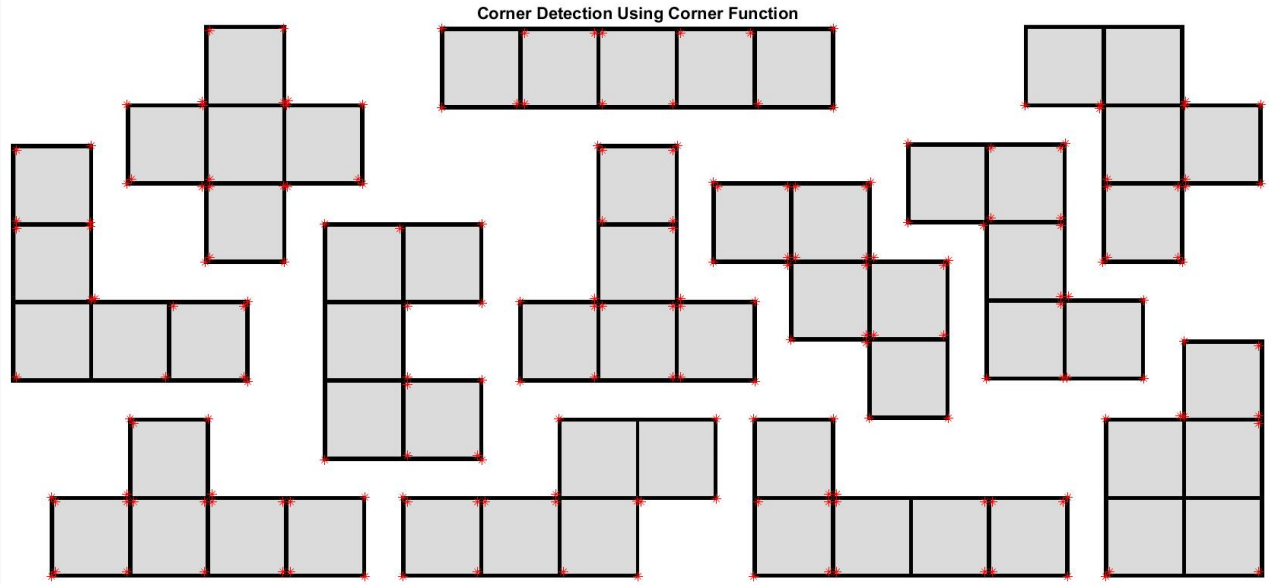
The corner function, as expected, highlights more corners in the image, with the MinimumEigenvalues function picking up the most corners. One would have expected that the corners of all the letters in “PUBLIC LIBRARY” would have been detected but none of the algorithms has a 100% detection rate in that regard.

Image 5: Some lego blocks

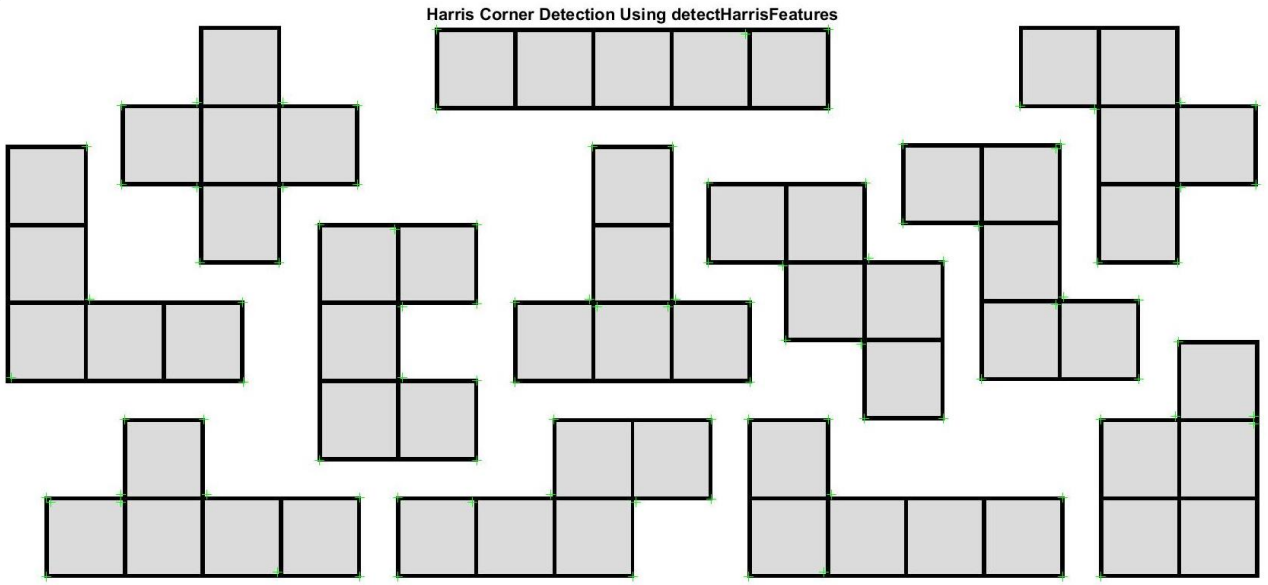


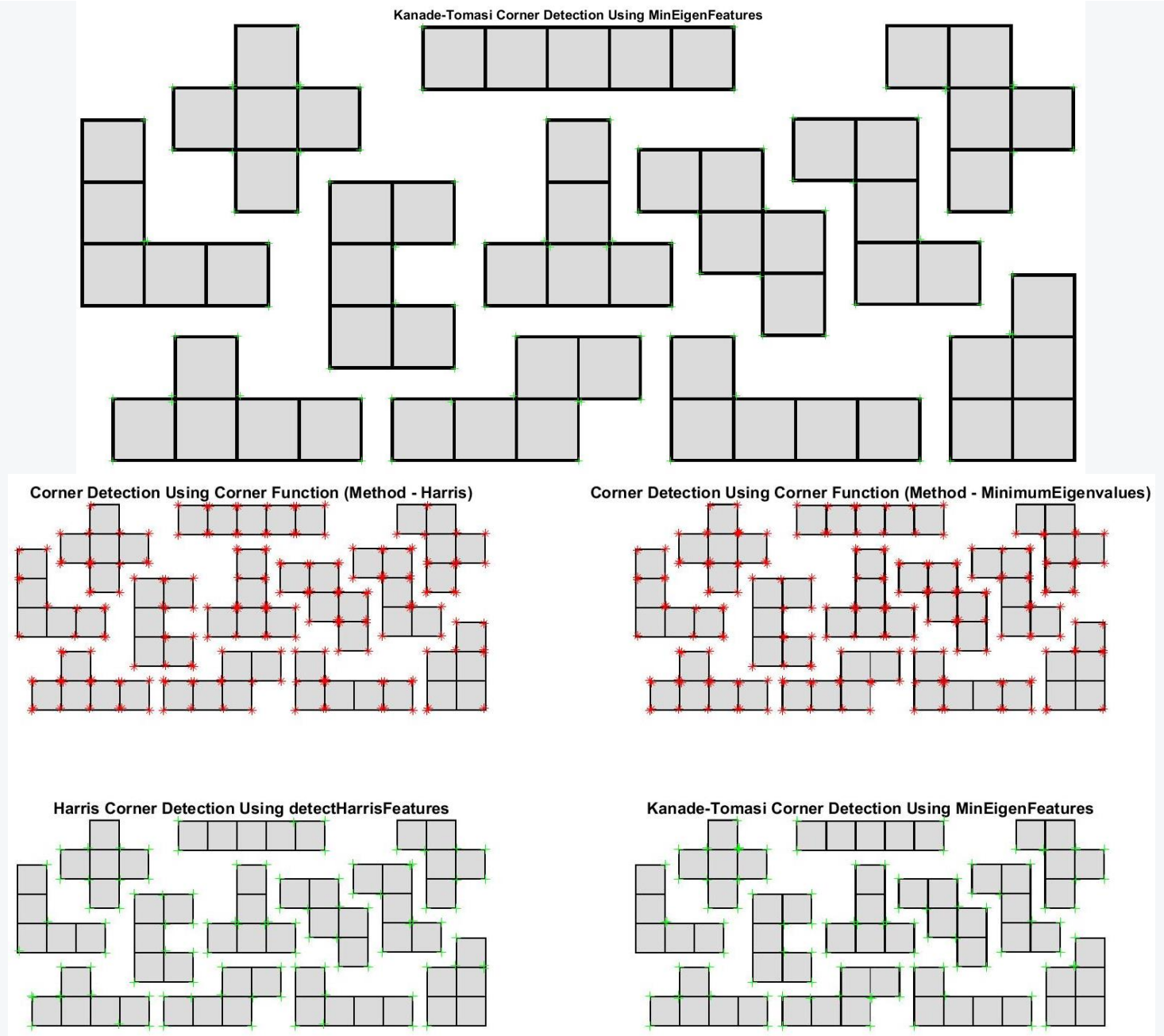


Method - Harris



Method - MinimumEigenvalues

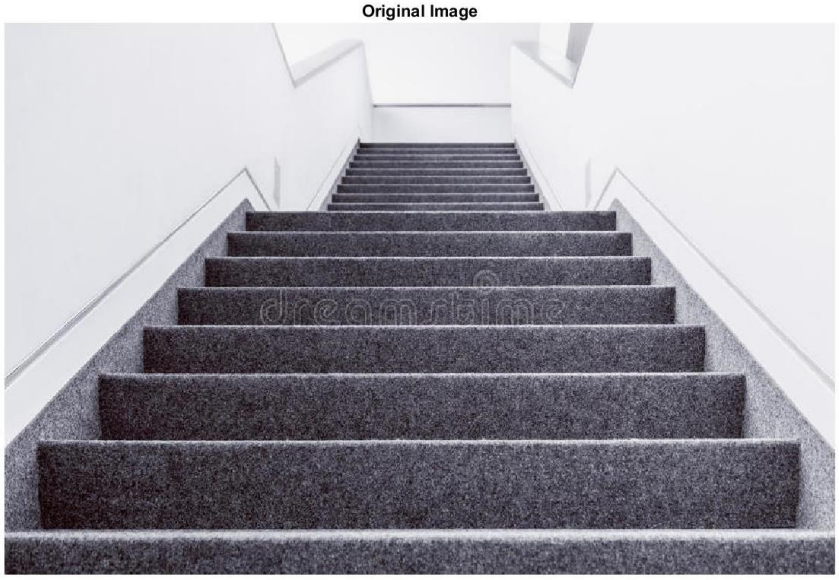




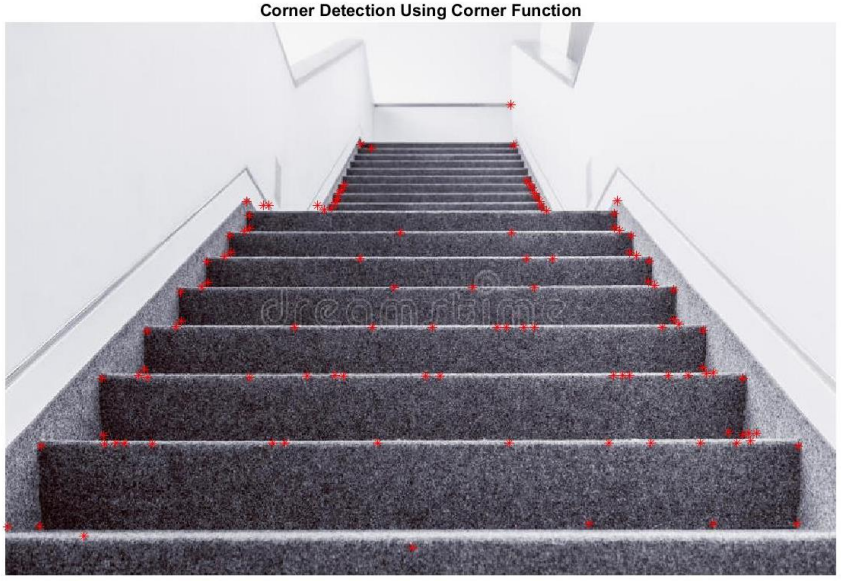
Just like in the edge detection section, this image really shows off the usefulness of a corner detector. In all four cases, we obtain good results. Notice how the corner function seems to overperform in certain areas by highlighting multiple neighbourhood points. **The functions from the computer vision toolbox, on the other hand, seem to have some nonmaximum suppression implemented**.

Image 6: A staircase

Original Image

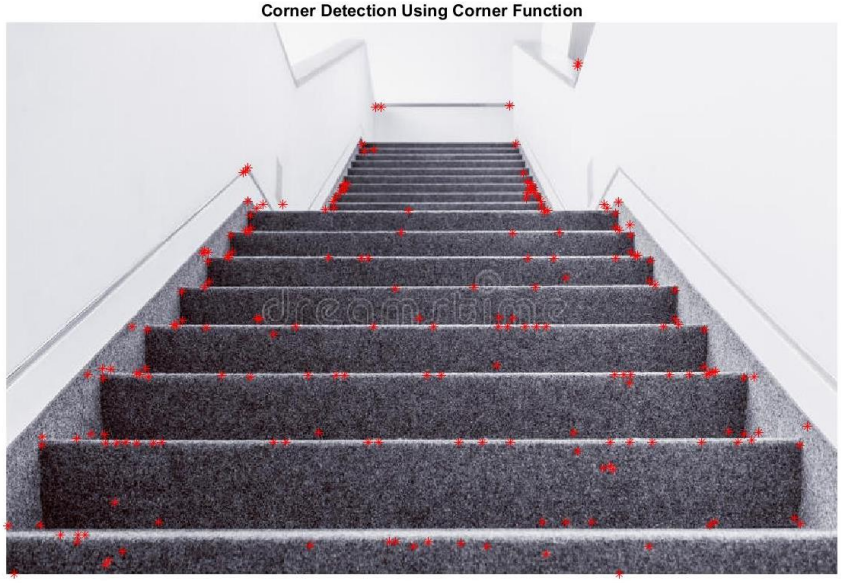


Corner Detection Using Corner Function



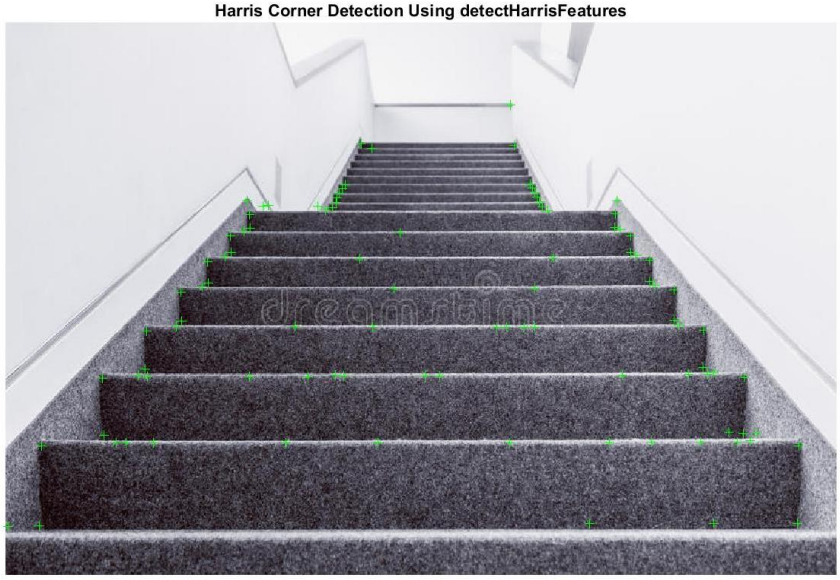
Method - Harris

Corner Detection Using Corner Function

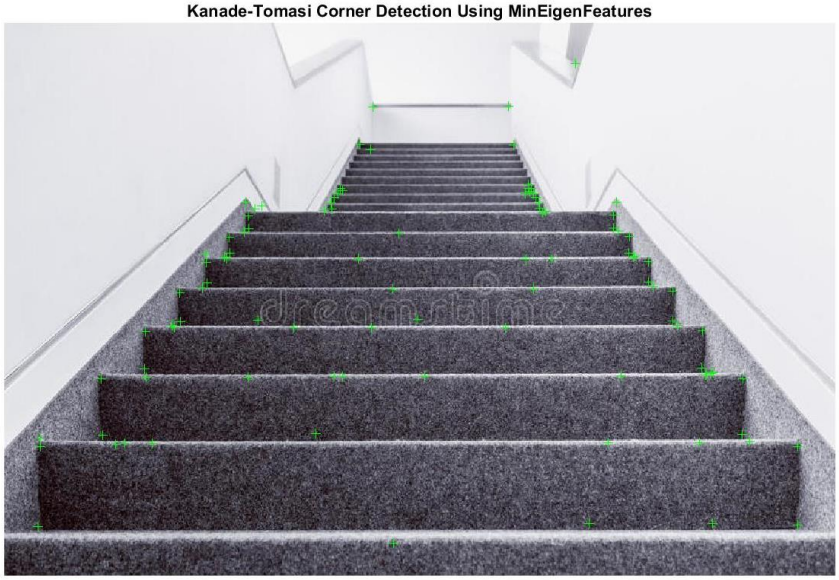


Method - MinimumEigenvalues

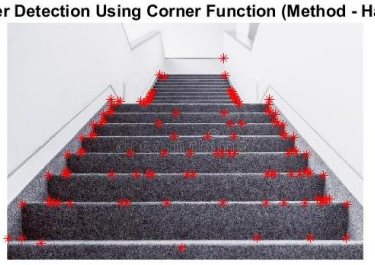
Harris Corner Detection Using detectHarrisFeatures



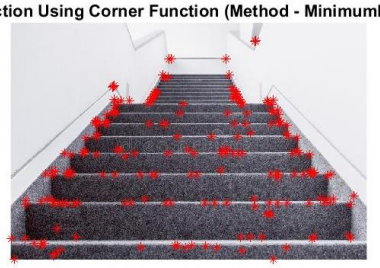
Kanade-Tomasi Corner Detection Using MinEigenFeatures



Corner Detection Using Corner Function (Method - Harris)

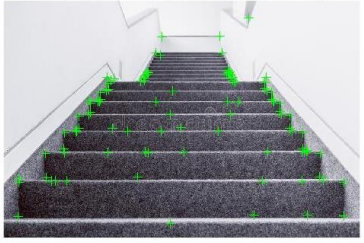


Corner Detection Using Corner Function (Method - MinimumEigenvalues)



Harris Corner Detection Using detectHarrisFeatures

Kanade-Tomasi Corner Detection Using MinEigenFeatures



With the above results, **we can immediately see the impact of object texture on image processing**. Due to the rough texture of the rug, unwanted areas are identified as corners.

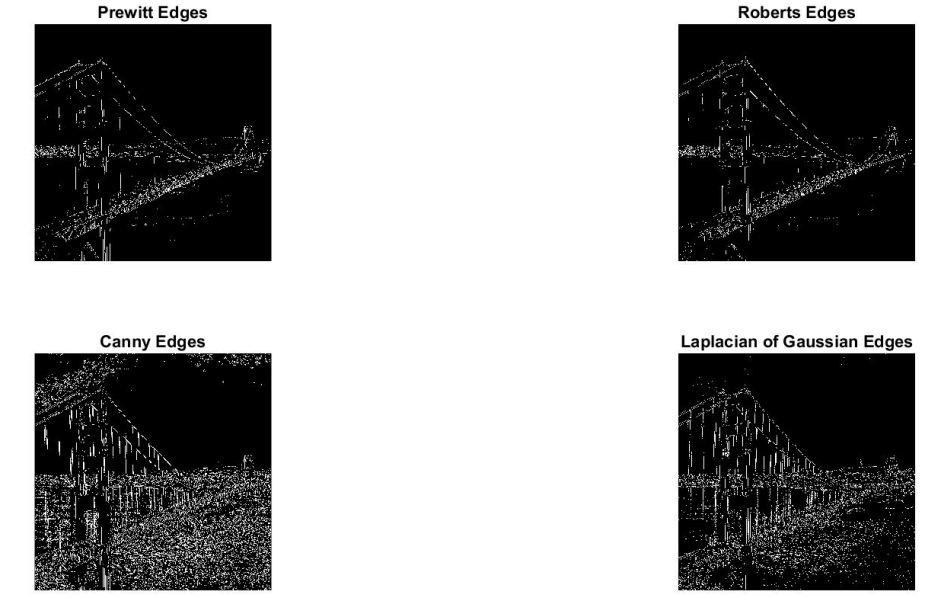
DISCUSSION

General Comments

In addition to the comments in the previous sections, the following could be noted:

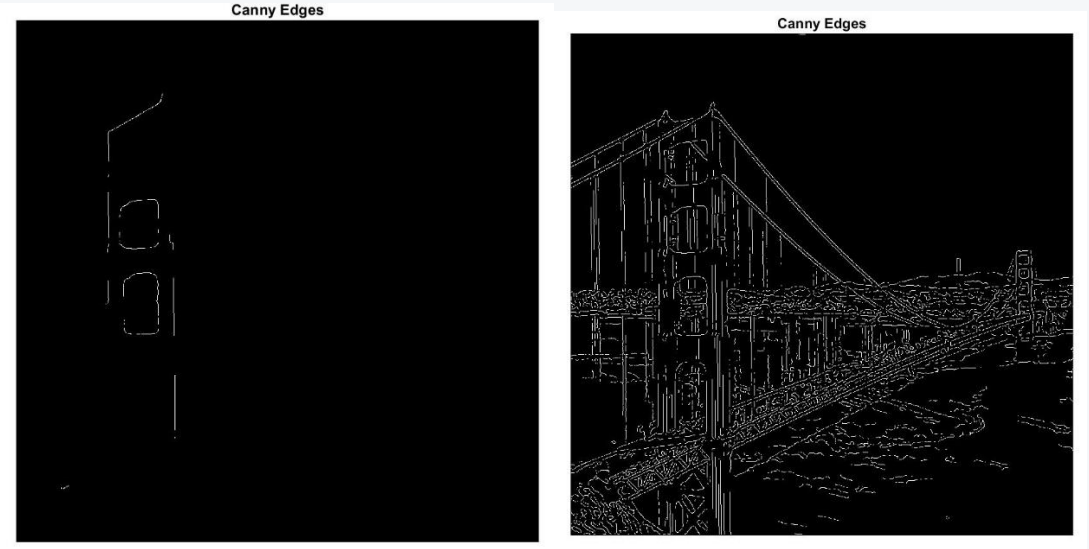
- In both cases, the images were smoothed using a Gauss filter before applying the algorithm to avoid obtaining **spurious corners and edges.**
- What happens if we try to detect the edges and corner without smoothing the images first?

- The results show many more detected points, some of which are bound to be spurious corners.



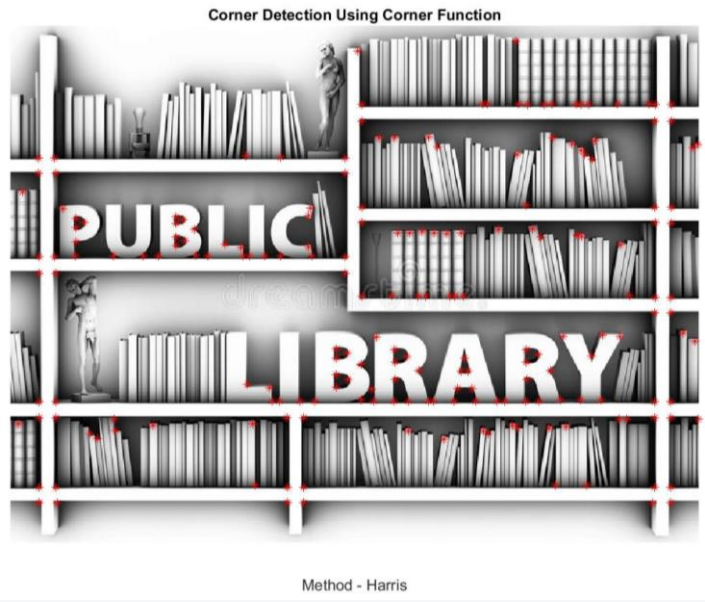
Since the Canny detector is really precise, it returns several spurious edges when we don't smoothen the input image.

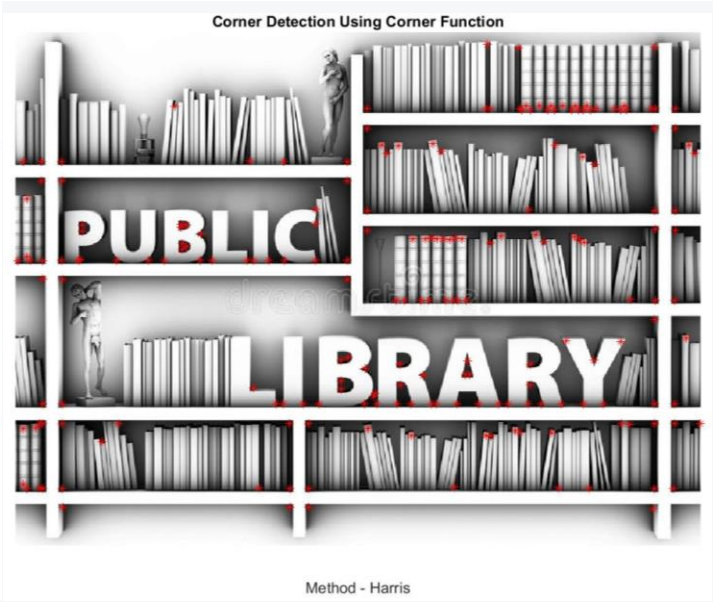
- The performance of different edge and corner detectors are partly dependent on the type of objects in the input image. Like we saw in the results, for objects that are not clearly defined (e.g., water, rug, etc.), the detection results are not so nice.
- For clearly defined objects like the lego blocks we saw, or a building, one can appreciate the outcome of edge and corner detection.
- Texture plays a major role in how good the result of an edge/corner detector will be.
- **Threshold matters!** By specifying the threshold for our detectors, we can fine-tune our results.



The above two images are the result of Canny edge detector on the same input image. The difference is that a 0.9 threshold within the [0 1] range is applied on the first image.

- When the threshold is not explicitly defined MATLAB assigns a default threshold.
- Adjusting the sensitivity in the Harris corner detection using the corner function alters the results. A higher sensitivity will return more corners. In the following, for the first image, a sensitivity of 0.01 was used, while 0.1 was used for the second image.





What's the computation time for the functions?
I used tic-toc to check which is faster to compute, and I obtained the following:

```
6    % Read the images whose edges wil
```

Command Window

New to MATLAB? See resources for Getting Started.

```
Elapsed time is 2.337093 seconds.
Elapsed time is 2.572480 seconds.
Elapsed time is 1.372932 seconds.
Elapsed time is 1.310669 seconds.
Elapsed time is 1.714405 seconds.
Elapsed time is 1.444907 seconds.
>>
```

The edge detection function has a time range of ~1.3 to 2.6 seconds. This is reflective of the size of the input image.

```
5
```

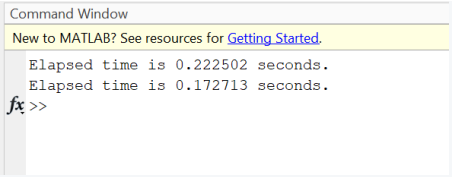
Command Window

New to MATLAB? See resources for Getting Started.

```
Elapsed time is 4.641893 seconds.
Elapsed time is 2.058390 seconds.
Elapsed time is 1.811803 seconds.
Elapsed time is 1.899219 seconds.
Elapsed time is 2.817299 seconds.
Elapsed time is 2.016938 seconds.
>>
```

- This is for the corner detection function.
- Generally, the corner detection function is more time-consuming.

--- I also used the tic-toc syntax to check which corner detector is faster, and I obtained the following:

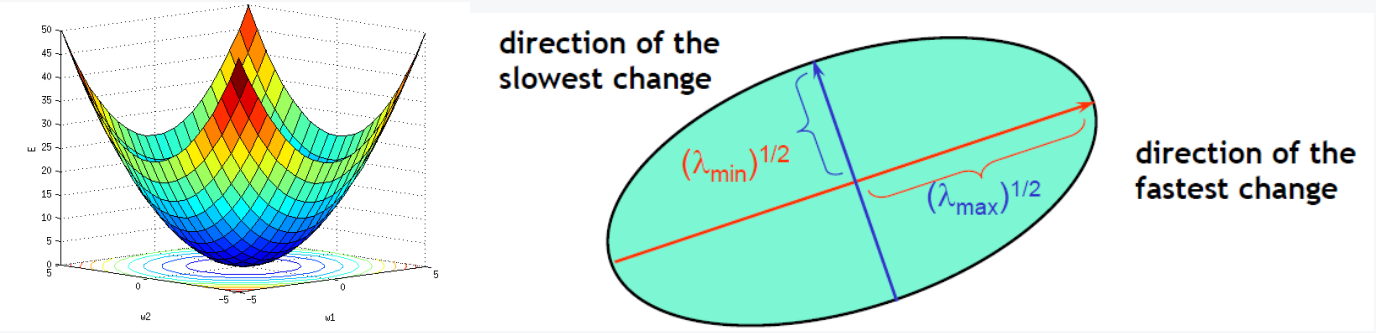


The first time is for the detectHarrisFeatures function, while the second is for the detectMinEigenFeatures.

- I must confess that I'm surprised that the Kanade-Tomasi algorithm was faster than the Harris algorithm. Ordinarily, I would have expected the reverse to be the case because of the need to compute a square root in the Kanade algorithm as seen below:

$$\lambda_{\pm} = \frac{1}{2}\left[(h_{11} + h_{22}) \pm \sqrt{4h_{12}h_{21} + (h_{11} - h_{22})^2}\right]$$

Additional Discussion



- As we can observe, it's in our best interest to heighten the response of our algorithm to corners by ensuring that we pinpoint area where the slightest move away result in massive variation in intensity. This can be achieved by thresholding the minimum eigenvalue since it represents the direction of slowest change.
- The threshold used is dependent both on the image and the corner detector used.

Generally,

- The Harris operator is computationally more advantageous than the Kanade-Tomasi method. This is predicated on the fact that in the calculation of f, there's no need to find the square root of any parameter. This greatly reduces the complexity of problems and puts significantly less strain on computational resources.

$$\lambda_{\pm} = \frac{1}{2}\left[(h_{11} + h_{22}) \pm \sqrt{4h_{12}h_{21} + (h_{11} - h_{22})^2}\right]$$

- This was particularly important around the 80's when computers with high computational power were particularly hard to come by.
- The threshold used is dependent both on the image and the corner detector used.

REFERENCES

Reinhard Klette. 2014. Concise Computer Vision: An Introduction into Theory and Algorithms. Springer Publishing Company, Incorporated.

https://en.wikipedia.org/wiki/Harris_corner_detector

https://3dwarehouse.sketchup.com/model/ue842f8d7-cf23-4cf5-8702-a7c917416363/6-Storey-Modern-Building?hl=tr

https://www.worldatlas.com/articles/the-most-famous-structures-in-the-world.html

https://www.huboo.com/blog/news/warehouse-the-latest-boom-in-the-real-estate-sector/

https://www.wallpaperbetter.com/en/search?q=Summer+Scenery

https://www.quora.com/How-many-different-shapes-can-be-made-with-10-2x3-LEGO-bricks-and-no-rotation-of-bricks-is-possible-Mathematically-how-do-you-setup-this-problem

https://www.dreamstime.com/stock-illustration-d-illustration-black-white-library-bookshelf-background-image91430672

https://www.dreamstime.com/stock-photo-modern-interior-empty-steps-low-angle-black-white-indoor-picture-image71505604

APPENDIX

Edge Detection

The single m-file for edge detection:

```
% Edge Detection

clear all
clc

% Read the images whose edges will be detected
img = imread('bridge.jpg');

% First of all, smoothen the image using MATLAB's inbuilt
Gaussian filter.
% In the discussion, some results were obtained with
applying Gaussian
% filtering first for the sake of comparison.
I = imgaussfilt(img, 3);

% The row, column, and channels of the image are obtained
along with the
% cardinality of the image.
[r, c, ch] = size(I);
Card = r*c;

% This is added in case the image introduced is an RGB
image.
% It functions to convert it to a gray-scale image.
if (ch == 3)
    I = rgb2gray(I);
end

% Find the Edges using Prewitt, Roberts, Canny, and log
% First Derivative Edge Detectors
P = edge(I, 'Prewitt');
R = edge(I, 'Roberts');
C = edge(I, 'Canny');
% Second Derivative Edge Detector
L = edge(I, 'log');

% Plots - Each image will be in a different figure
figure %1 - Original Image
```

```
imshow(img)
title('Original Image')

figure %2
imshow(P)
title('Prewitt Edges')

figure %3
imshow(R)
title('Roberts Edges')

figure %3
imshow(C)
title('Canny Edges')

figure %4
imshow(L)
title('Laplacian of Gaussian Edges')

% Subplots are used to place the above results on the same
figure for the
% sake of comparison.
figure %5
subplot(2,2,1)
imshow(P)
title('Prewitt Edges')

subplot(2,2,2)
imshow(R)
title('Roberts Edges')

subplot(2,2,3)
imshow(C)
title('Canny Edges')

subplot(2,2,4)
imshow(L)
title('Laplacian of Gaussian Edges')
```

Corner Detection

The single m-file for corner detection:

```
% Corner Detection

clear all
clc

% Read the images whose edges will be detected
img = imread('bridge.jpg');

% First of all, smoothen the image using MATLAB's inbuilt
Gaussian filter.
% In the discussion, some results were obtained with
applying Gaussian
% filtering first for the sake of comparison.
I = imgaussfilt(img, 3);

% The row, column, and channels of the image are obtained
along with the
% cardinality of the image.
[r, c, ch] = size(I);
Card = r*c;

% This is added in case the image introduced is an RGB
image.
% It functions to convert it to a gray-scale image.
if (ch == 3)
    I = rgb2gray(I);
end

% Detect the Corners
% Using Corner function
C_H = corner(I, 'Harris');
C_M = corner(I, 'MinimumEigenvalue');
% Using Functions from the Computer Vision Toolbox
H = detectHarrisFeatures(I);
ME = detectMinEigenFeatures(I);

% Plots - Each image will be in a different figure
```

```
figure %1 - Original Image
imshow(img)
title('Original Image')

figure %2
imshow(img)
hold on
plot(C_H(:,1),C_H(:,2),'r*');
title('Corner Detection Using Corner Function')
xlabel('Method - Harris')

figure %3
imshow(img)
hold on
plot(C_M(:,1),C_M(:,2),'r*');
title('Corner Detection Using Corner Function');
xlabel('Method - MinimumEigenvalues')

figure %3
imshow(img)
hold on;
plot(H.selectStrongest(50));
title('Harris Corner Detection Using detectHarrisFeatures')

figure %4
imshow(img)
hold on;
plot(ME.selectStrongest(50));
title('Kanade-Tomasi Corner Detection Using
MinEigenFeatures')

% Subplots are used to place the above results on the same
figure for the
% sake of comparison.
figure %6
subplot(2,2,1)
imshow(img)
hold on
plot(C_H(:,1),C_H(:,2),'r*');
title('Corner Detection Using Corner Function (Method -
Harris)')
```

```
subplot(2,2,2)
imshow(img)
hold on
plot(C_M(:,1),C_M(:,2),'r*');
title('Corner Detection Using Corner Function (Method -
MinimumEigenvalues)');

subplot(2,2,3)
imshow(img)
hold on;
plot(H.selectStrongest(100));
title('Harris Corner Detection Using detectHarrisFeatures')

subplot(2,2,4)
imshow(img)
hold on;
plot(ME.selectStrongest(100));
title('Kanade-Tomasi Corner Detection Using
MinEigenFeatures')
```