Assignment 4 (60/40 points)

Edge Detection (10)

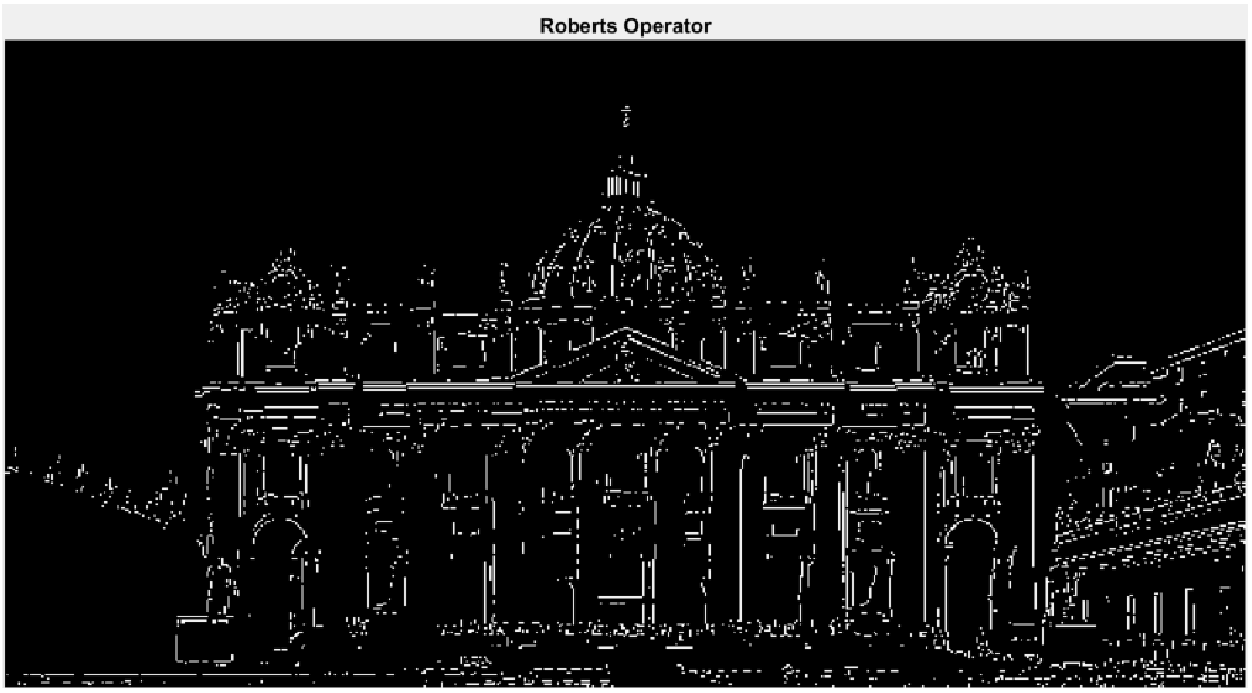
Use the `edge` function to generate results for Roberts, Canny, Sobel, and Prewitt operators on an image of your choice. Note also that the various edge functions support a number of parameters – feel free to explore those to get more interesting results. State which operator gives the best performance and why you think so. Notice also how this problem is different than the one using filters that I asked you to do for Assignment 3.

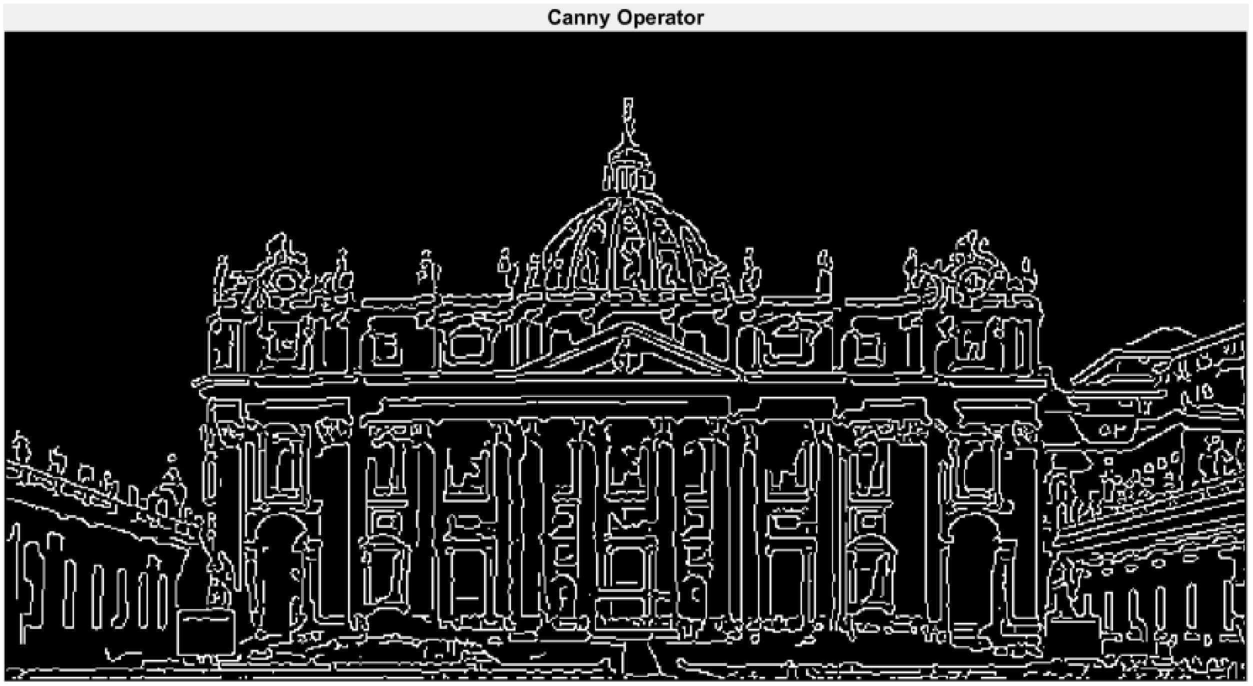
CODE:

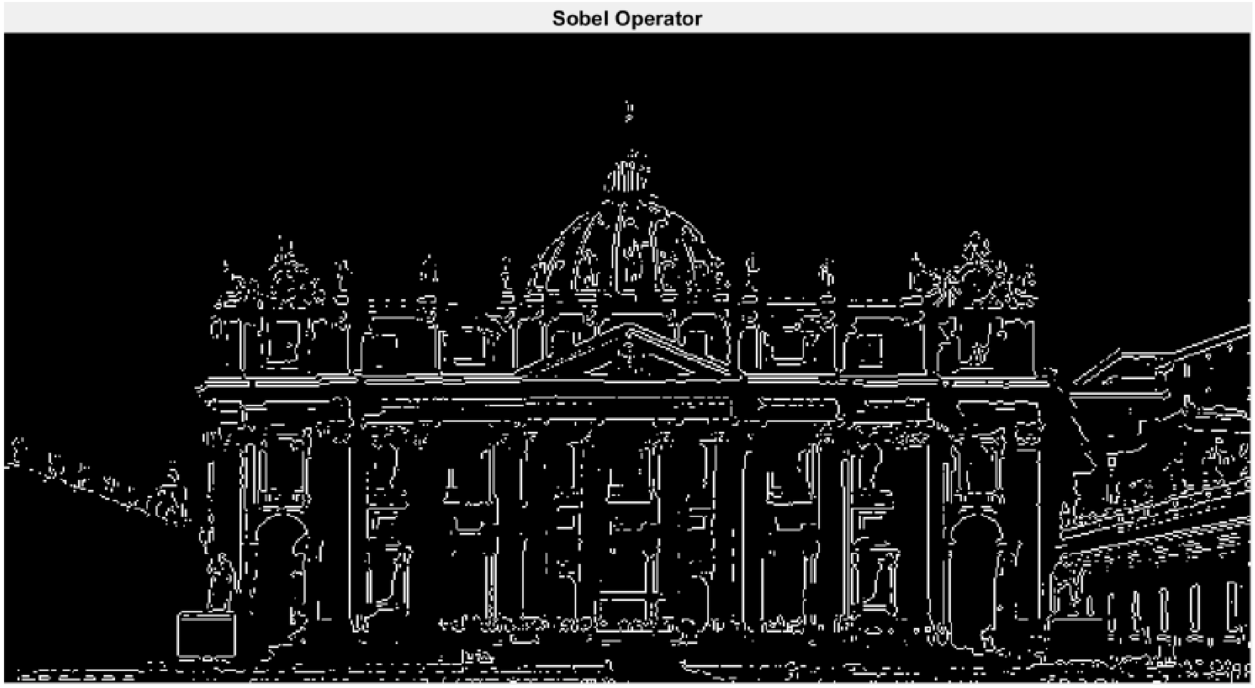
```
OG_Imgae = imread('Vatican.jpg');
OG_Imgae = rgb2gray(OG_Imgae);

figure;
imshow(OG_Imgae);
title('Original Image');

% Roberts operator
roberts_im = edge(OG_Imgae, 'Roberts');

% Canny operator
canny_im = edge(OG_Imgae, 'Canny');

% Sobel operator
sobel_im = edge(OG_Imgae, 'Sobel');

% Prewitt operator
prewitt_im = edge(OG_Imgae, 'Prewitt');

figure;
imshow(roberts_im);
title('Roberts Operator');

figure;
imshow(canny_im);
title('Canny Operator');

figure;
imshow(sobel_im);
title('Sobel Operator');

figure;
imshow(prewitt_im);
title('Prewitt Operator');
```

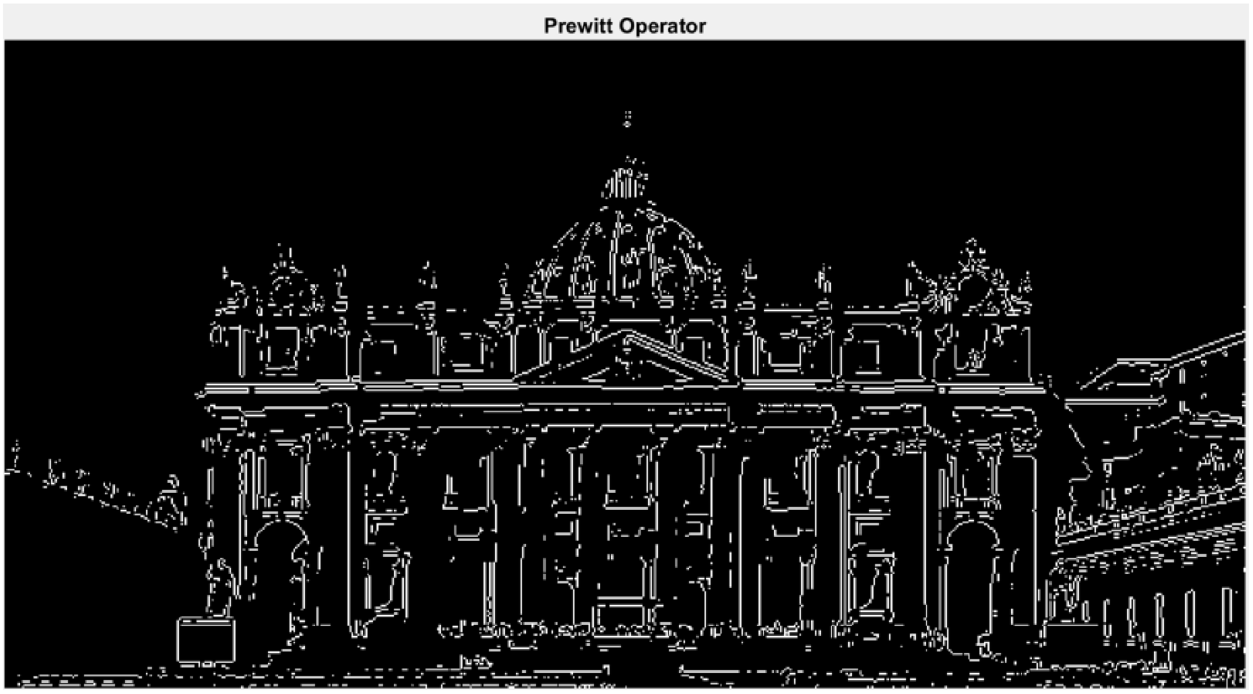
O/P:

Original Image


Roberts Operator


Canny Operator


Sobel Operator



Edge Filter (10)

Design a 7x7 “Sobel” operator and filter your image from the “Edge Detection” task above with your filter. The main idea behind the design of a proper Sobel-ish operator is to model the Gaussian derivate in one direction and the Gaussian in the perpendicular direction:

$$\frac{\partial G(x,y,\sigma)}{\partial x} = -\frac{x}{2\pi\sigma^4}e^{-\frac{\left(x^2+y^2\right)}{2\sigma^2}}$$

In this formulation, σ becomes a parameter of the filter; you can choose whatever you like, although choosing 1 makes the math much simpler. *x* and *y* are the distance from the center pixel of your filter. Aside from the size of your filter, how does it differ from the standard Sobel operator? How are the image results different than what you saw applying the standard Sobel operator?

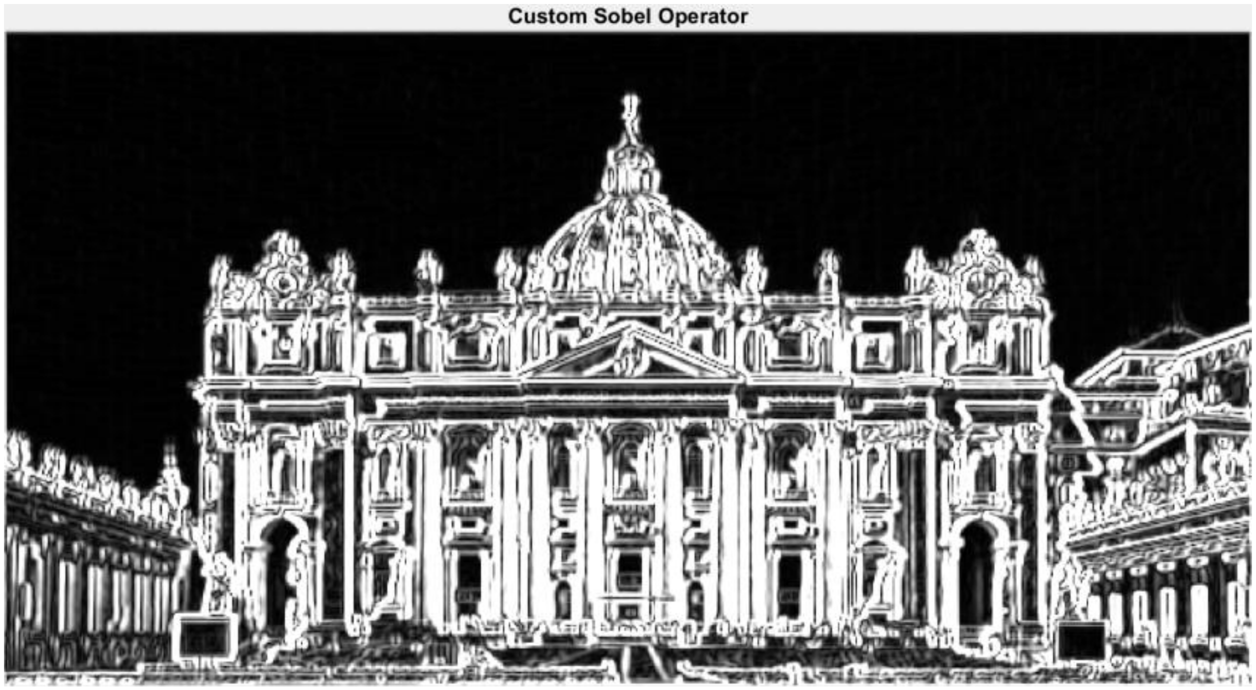
In this script, an input grayscale image is processed using a customized Sobel-like operator. The operator is tailored with unique sigma values, sigma_x and sigma_y, dictating the extent of smoothing in the horizontal and vertical orientations. The operator's elements are determined using the Gaussian derivative formula with these sigma values. Subsequently, the image undergoes separate filtering with the custom operator's horizontal and vertical components, followed by the computation of the gradient magnitude. The script then displays the original grayscale image and the filtered result, showcasing the gradient magnitude that accentuates edges.

Contrasted with the conventional Sobel operator, the distinctive feature of the custom approach lies in its employment of variable sigma values, enabling diverse levels of smoothing along horizontal and vertical axes. Unlike the standard Sobel operator, which utilizes a fixed kernel without Gaussian smoothing, the custom operator's adaptability to image nuances and noise levels results in varying edge thickness and appearance. This adaptability renders it more versatile for specific edge detection tasks.

CODE:

```
OG_image = imread('Vatican.jpg');
OG_image = rgb2gray(OG_image);

% Sigma Values
sigma_x = 1.5;
sigma_y = 1.0;

[x, y] = meshgrid(-3:3, -3:3);
K_X = -(x) .* exp(-((x.^2 + y.^2) / (2 * sigma_x^2)));
K_Y = -(y) .* exp(-((x.^2 + y.^2) / (2 * sigma_y^2)));

% Filter the image with the custom Sobel-like operator
sobel_x = conv2(double(OG_image), K_X, 'same');
sobel_y = conv2(double(OG_image), K_Y, 'same');

% Calculate the gradient magnitude
gradient_magnitude = sqrt(sobel_x.^2 + sobel_y.^2);


figure;
subplot(1, 2, 1);
imshow(OG_image);
title('Original Image');

subplot(1, 2, 2);
imshow(uint8(gradient_magnitude));
title('Custom Sobel Operator');
```

O/P:



Histogram-based segmentation (20)

Implement histogram based segmentation on your image as follows:

a) Show your image.

b) Display the histogram and identify the peaks of your histogram with the “objects” that they correspond to.

c) Specify the ranges that you will use to identify the binary objects.

d) Show the identified objects as binary images for each range. (Remember to scale the images for display so that objects can be seen.)

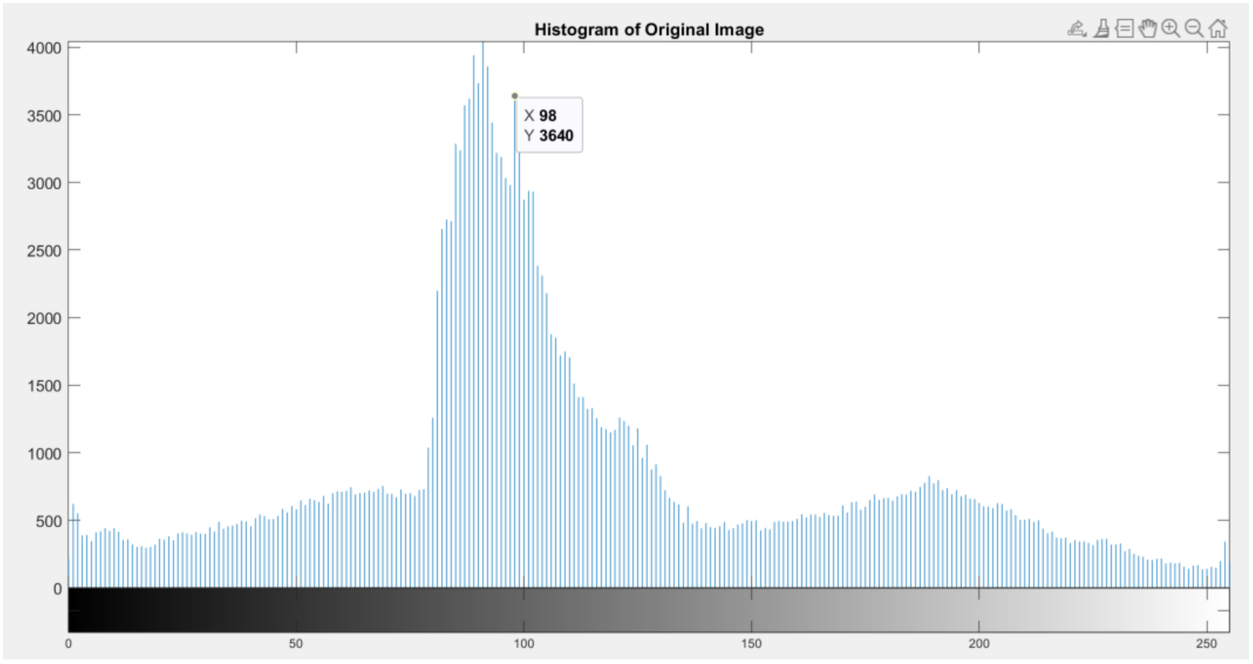
e) Finally construct the histogram-based segmented image, by combining the binary images.

CODE:

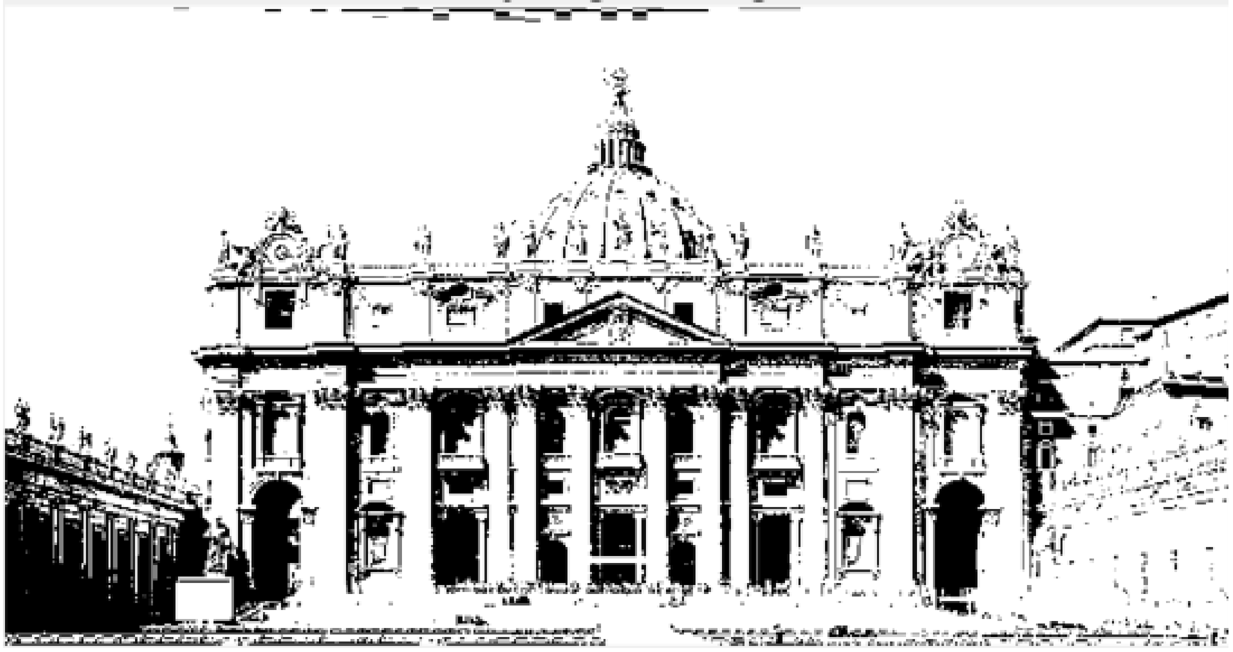
```
% a) Show your image.
image = imread('Vatican.jpg');
OG_image = rgb2gray(image);

figure;
imshow(OG_image);
title('Original Image');

% b) Display the histogram and identify the peaks of your histogram with the
"objects” that they correspond to.
figure;
imhist(OG_image);
title('Histogram of Original Image');

% c) Specify the ranges that you will use to identify the binary objects.
% [80,1050] to [130,850] x range from 80 to 130 holds the maximum range of peaks
% Let us consider different ranges under this

%d) Show the identified objects as binary images:
% Create binary masks for each specified range
range1 = [80, 1050];
range2 = [100, 600];
range3 = [120, 400];
range4 = [90, 750];
range5 = [110, 500];

% Create binary images for each specified range
binaryImage1 = (OG_image >= range1(1)) & (OG_image <= range1(2));
binaryImage2 = (OG_image >= range2(1)) & (OG_image <= range2(2));
binaryImage3 = (OG_image >= range3(1)) & (OG_image <= range3(2));
binaryImage4 = (OG_image >= range4(1)) & (OG_image <= range4(2));
binaryImage5 = (OG_image >= range5(1)) & (OG_image <= range5(2));

% Display the binary images
figure;
subplot(2, 3, 1);
imshow(binaryImage1);
title('Binary Image for Range 1');
```

```
subplot(2, 3, 2);
imshow(binaryImage2);
title('Binary Image for Range 2');

subplot(2, 3, 3);
imshow(binaryImage3);
title('Binary Image for Range 3');

subplot(2, 3, 4);
imshow(binaryImage4);
title('Binary Image for Range 4');

subplot(2, 3, 5);
imshow(binaryImage5);
title('Binary Image for Range 5');

% e) Finally construct the histogram-based segmented image, by combining the binary
images.
segmentedImage = binaryImage1 | binaryImage2 | binaryImage3 | binaryImage4 |
binaryImage5;

subplot(2, 3, 6);
imshow(segmentedImage);
title('Segmented Image');
```

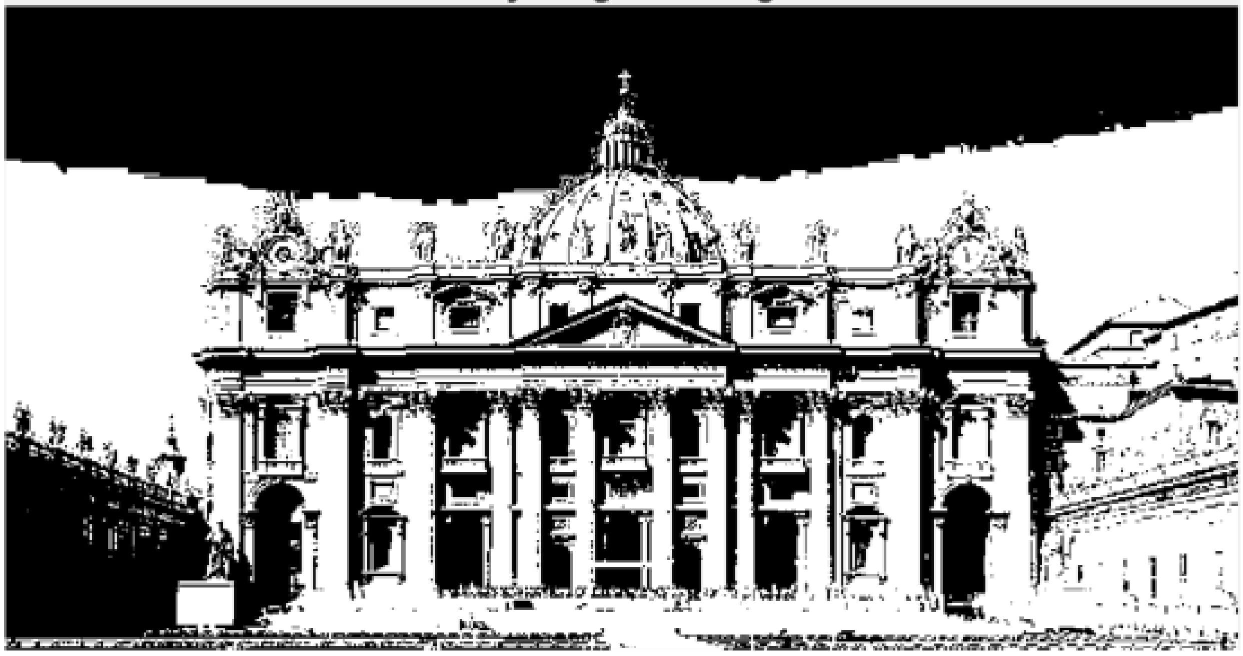
O/P:

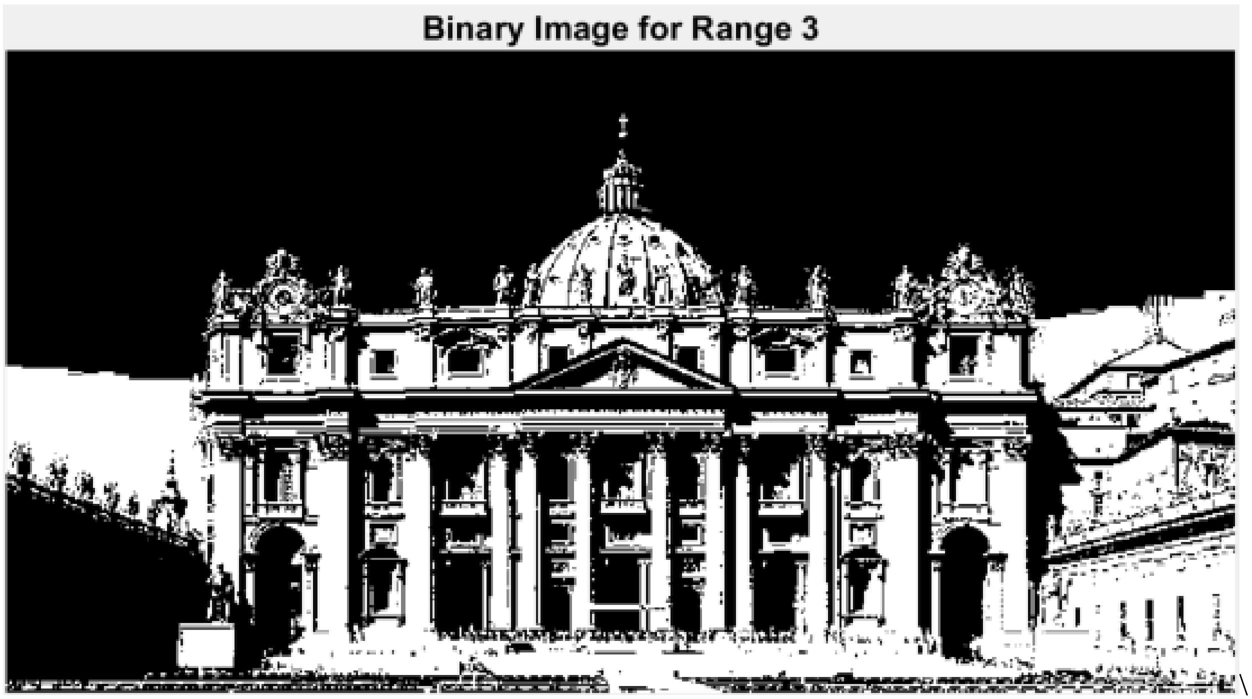


Binary Image for Range 1

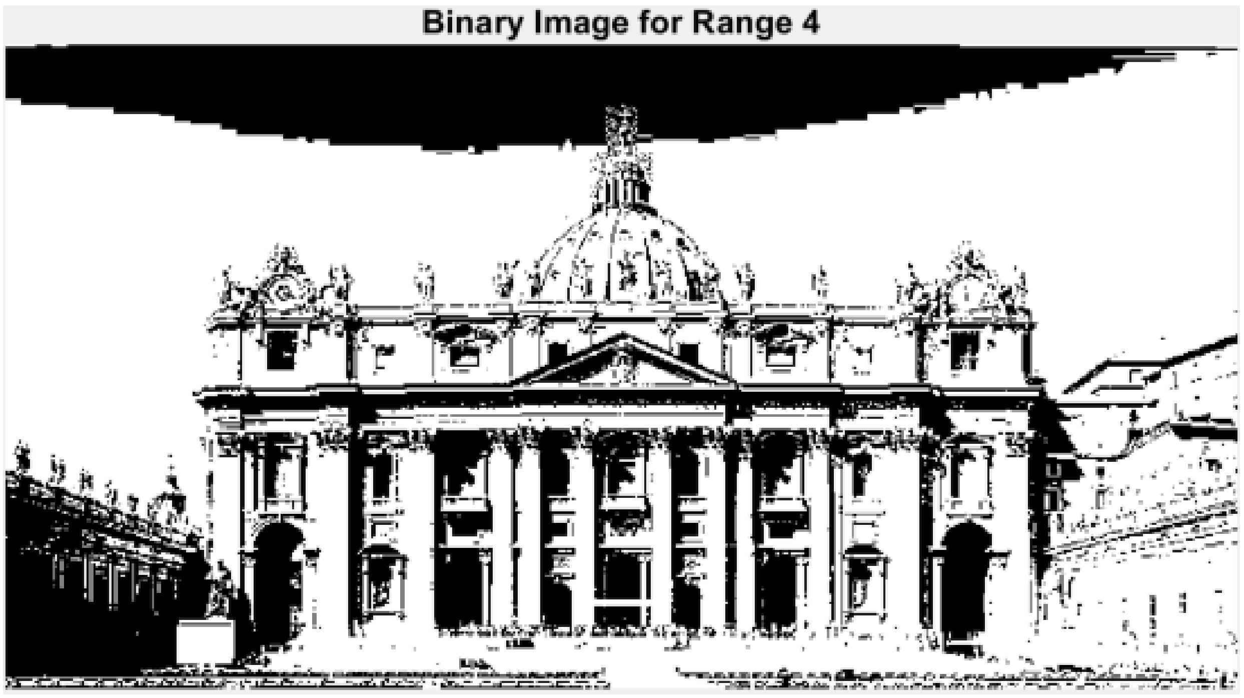


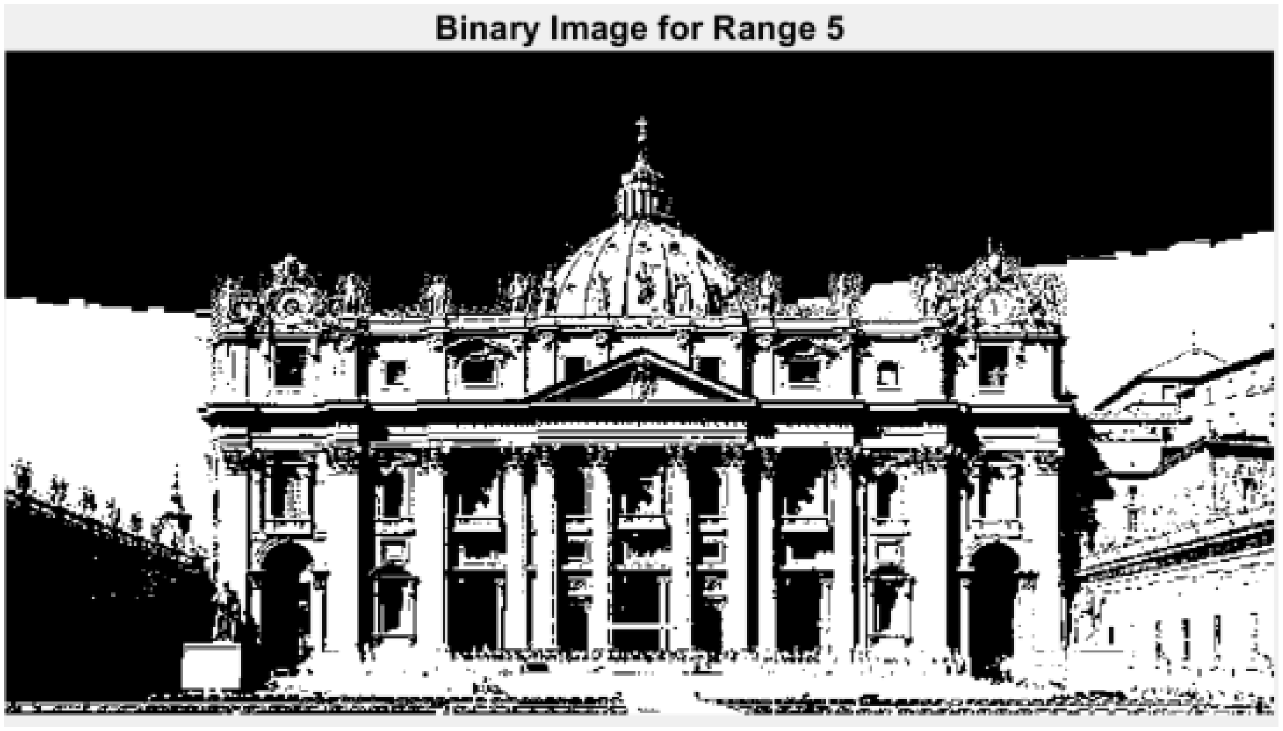
Binary Image for Range 2

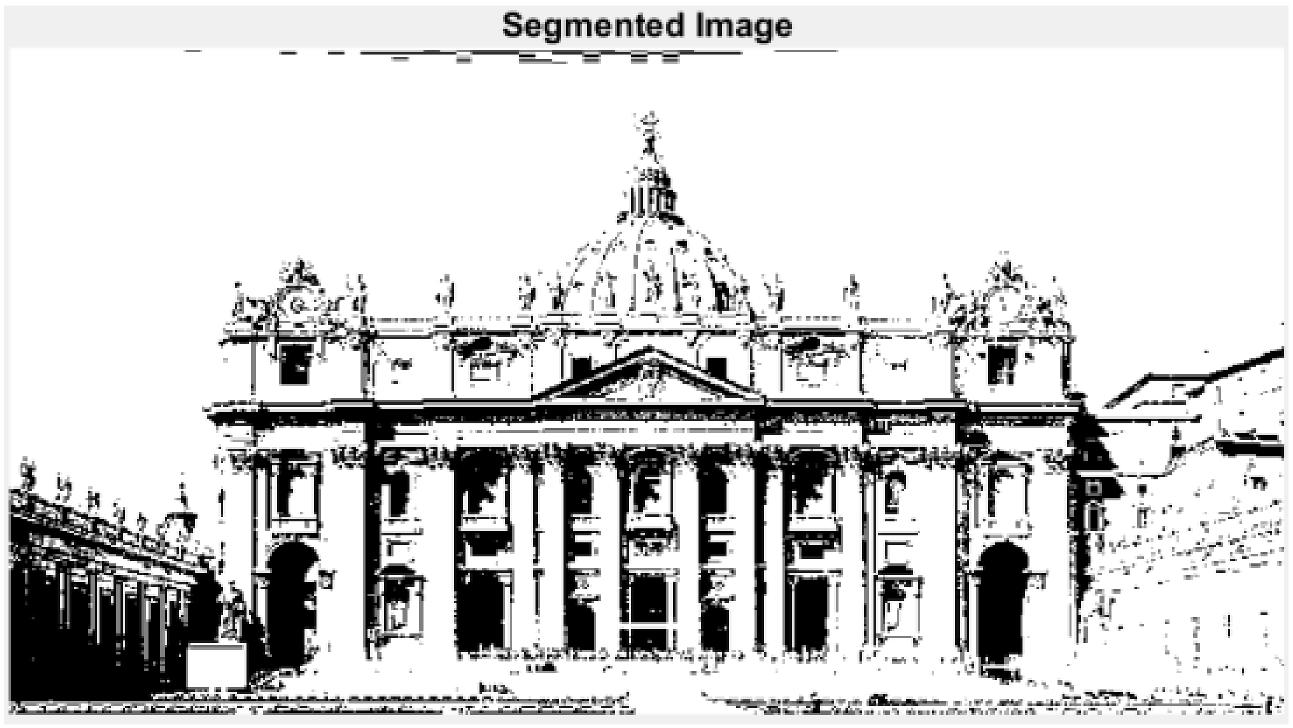



Binary Image for Range 3

\


Binary Image for Range 4


Binary Image for Range 5


Segmented Image

481 Students: Noise reduction *(*20/0)

a) On an image of your choice, use the `imnoise` function to generate two noise corrupted images as follows: (the article at https://stackoverflow.com/questions/14435632/impulse-gaussian-and-salt-and-pepper-noise-with-opencv show one way to do this with Python and openCV)
`noise_gaussian =imnoise(Image,’gaussian’,0,0.05);`
`noise_saltAndpepper=imnoise(Image,’salt & pepper’, 0.02);`

b) Use `subplot` to display the original image and the two noise corrupted images.

c) Use the function `fspecial` to design averaging filters of size (3x3), (5,5), and (7x7). Use `subplot` to display the `noise_saltAndpepper` image and the three averaged filtered results. Do the same for the `noise_gaussian` image.

d) Use the `medfilt2` function to perform median filtering on the `noise_saltAndpepper` image. Design the median filters to work with window sizes of (3x3), (5x5), and (7x7). Use your filters on the `noise_gaussian` image also and display as in part c).

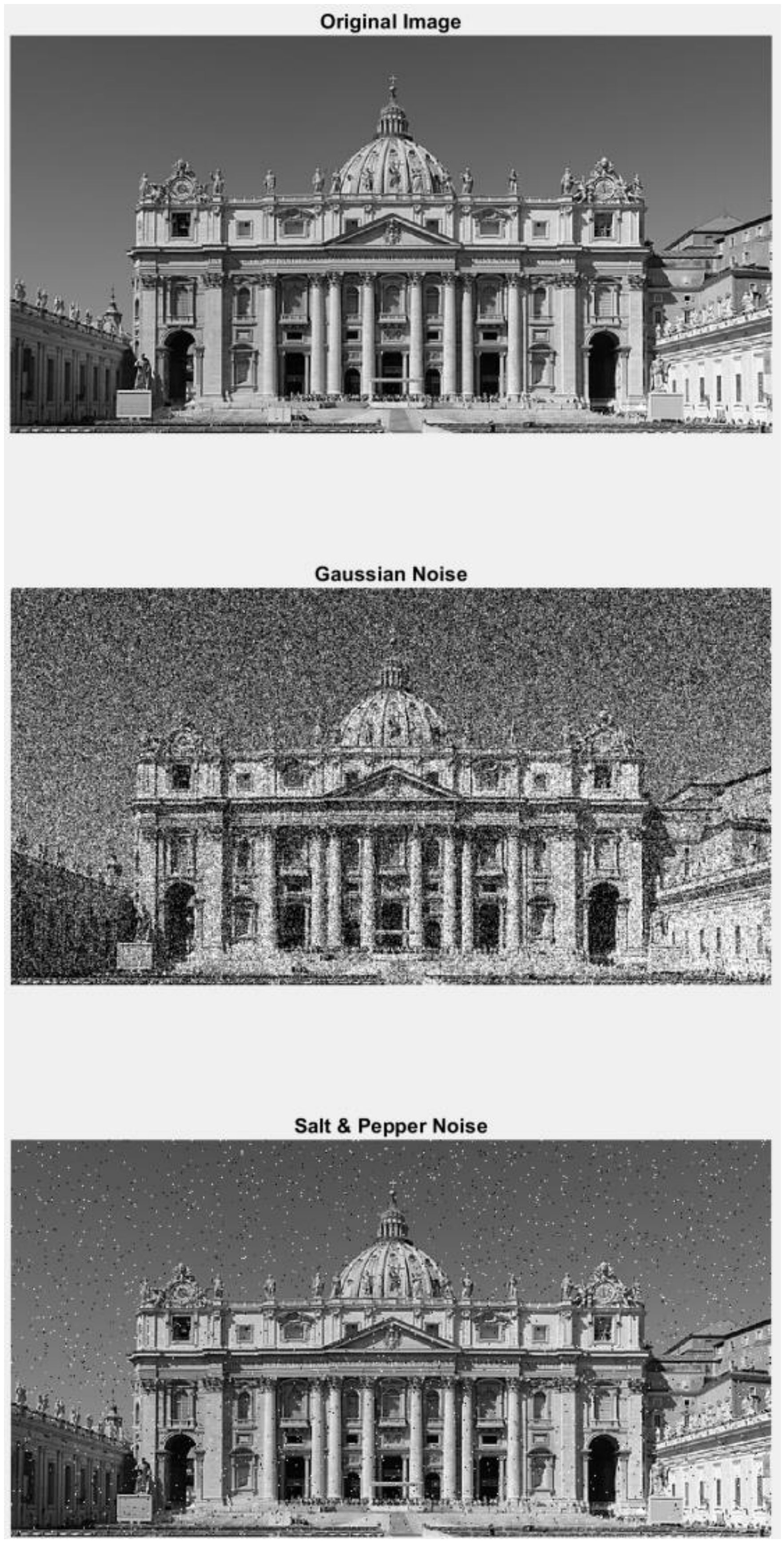
CODE:

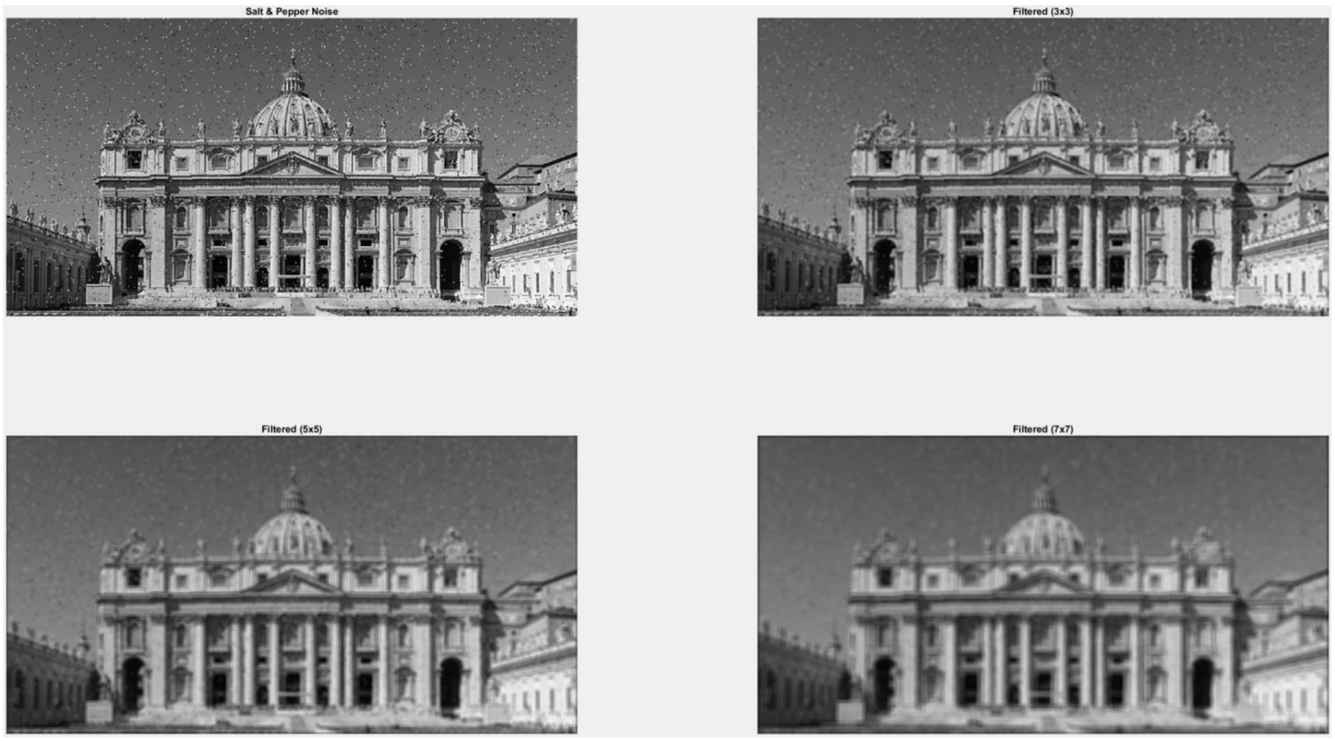
```
% a) On an image of your choice, use the imnoise function to generate two noise
corrupted images
OG_Image = imread('Vatican.jpg');
OG_Image = rgb2gray(OG_Image);
% Noise image 1
noise_1 = imnoise(OG_Image, 'gaussian', 0, 0.05);
% Noise Image 2
noise_2 = imnoise(OG_Image, 'salt & pepper', 0.02);

% b) Use subplot to display the original image and the two noise corrupted images.
figure;
subplot(1, 3, 1);
imshow(OG_Image);
title('Original Image');

subplot(1, 3, 2);
imshow(noise_1);
title('Gaussian Noise');

subplot(1, 3, 3);
imshow(noise_2);
title('Salt & Pepper Noise');

% c) design averaging filters of size (3x3), (5,5), and (7x7)
filter_3x3 = fspecial('average', [3 3]);
filter_5x5 = fspecial('average', [5 5]);
filter_7x7 = fspecial('average', [7 7]);

SP_3x3 = imfilter(noise_2, filter_3x3);
SP_5x5 = imfilter(noise_2, filter_5x5);
SP_7x7 = imfilter(noise_2, filter_7x7);

G_3x3 = imfilter(noise_1, filter_3x3);
```

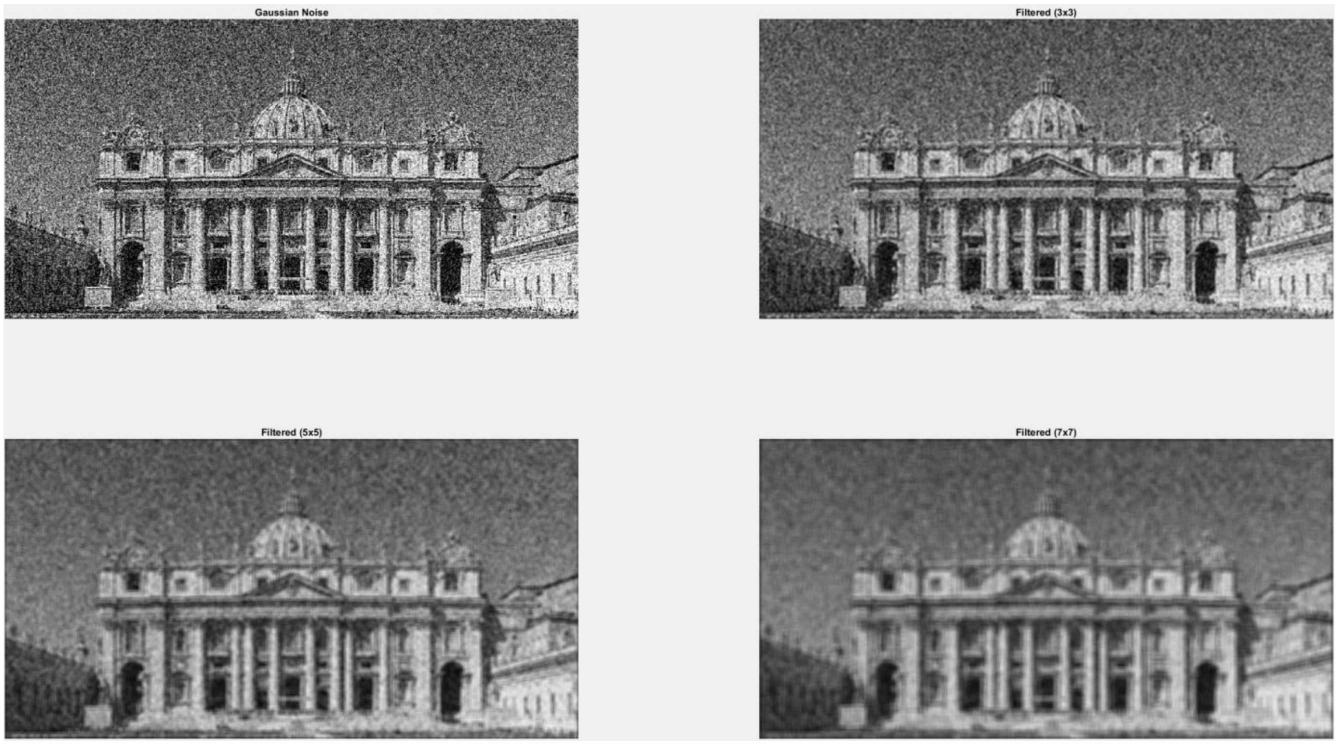
```
G_5x5 = imfilter(noise_1, filter_5x5);
G_7x7 = imfilter(noise_1, filter_7x7);

figure;
subplot(2, 2, 1);
imshow(noise_2);
title('Salt & Pepper Noise');

subplot(2, 2, 2);
imshow(SP_3x3);
title('Filtered (3x3)');

subplot(2, 2, 3);
imshow(SP_5x5);
title('Filtered (5x5)');

subplot(2, 2, 4);
imshow(SP_7x7);
title('Filtered (7x7)');

figure;
subplot(2, 2, 1);
imshow(noise_1);
title('Gaussian Noise');

subplot(2, 2, 2);
imshow(G_3x3);
title('Filtered (3x3)');

subplot(2, 2, 3);
imshow(G_5x5);
title('Filtered (5x5)');

subplot(2, 2, 4);
imshow(G_7x7);
title('Filtered (7x7)');

% d) Perform Median Filtering on Noisy Images:

median_SP_3x3 = medfilt2(noise_2, [3 3]);
median_SP_5x5 = medfilt2(noise_2, [5 5]);
median_SP_7x7 = medfilt2(noise_2, [7 7]);

median_G_3x3 = medfilt2(noise_1, [3 3]);
median_G_5x5 = medfilt2(noise_1, [5 5]);
median_G_7x7 = medfilt2(noise_1, [7 7]);

figure;
subplot(2, 2, 1);
imshow(noise_2);
title('Salt & Pepper Noise');

subplot(2, 2, 2);
imshow(median_SP_3x3);
title('Median Filtered (3x3)');
```

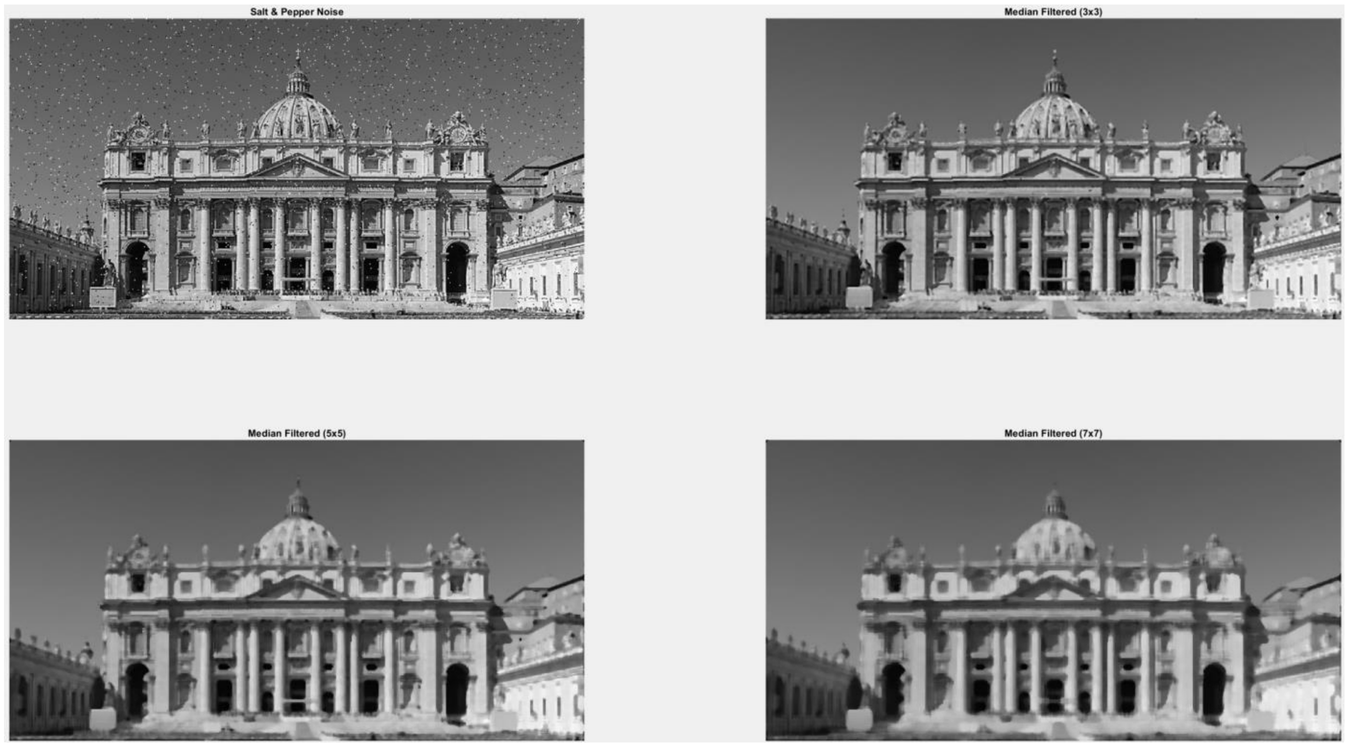
```
subplot(2, 2, 3);
imshow(median_SP_5x5);
title('Median Filtered (5x5)');

subplot(2, 2, 4);
imshow(median_SP_7x7);
title('Median Filtered (7x7)');

figure;
subplot(2, 2, 1);
imshow(noise_1);
title('Gaussian Noise');

subplot(2, 2, 2);
imshow(median_G_3x3);
title('Median Filtered (3x3)');

subplot(2, 2, 3);
imshow(median_G_5x5);
title('Median Filtered (5x5)');

subplot(2, 2, 4);
imshow(median_G_7x7);
title('Median Filtered (7x7)');
```

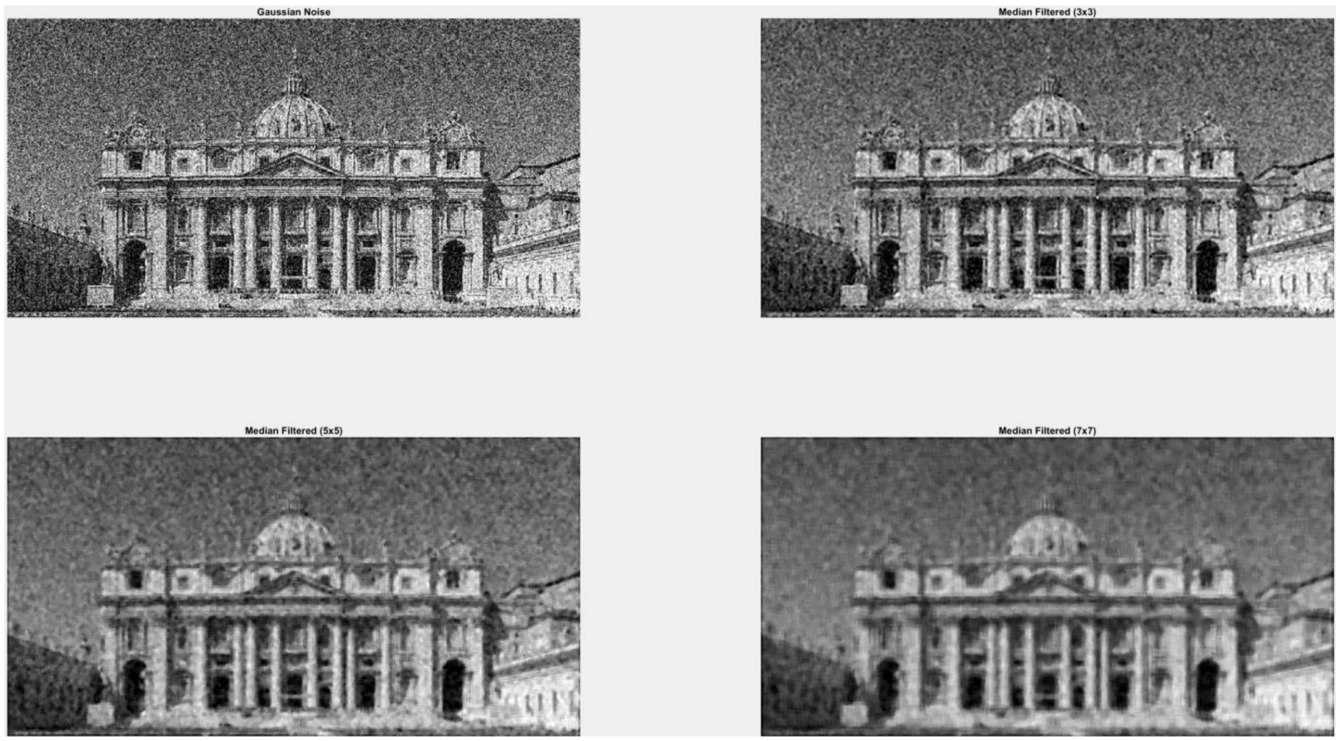
O/P:


Original Image
Gaussian Noise
Salt & Pepper Noise


Salt & Pepper Noise
Filtered (3x3)
Filtered (5x5)
Filtered (7x7)


Gaussian Noise
Filtered (3x3)
Filtered (5x5)
Filtered (7x7)





General submission instructions:

(a) Be kind to your aging, over-worked professor and submit only a single document. This can be pdf, MS Word, OpenOffice, etc. Do not submit a zip file.

(b) Your single document should include the input image for your problem, if required, and answers to each of the sub-problems (text, image or both, as appropriate). Your document should also include code that you wrote to generate your answers.

(c) You may use any images you like for the programming; I encourage you to use images that might be useful/interesting for your final project.

(d) Feel free to use whatever functions MatLab supplies (unless otherwise specified). Also feel free to write your own, if you are so inclined; it will take more time, but you will gain a deeper understanding of the material.

(e) Point values for each question are indicated as ***(x/y)*** in which ***x*** is the point value for 481 students and ***y*** is the point value for 381 students.