Fall 2019

Computer Vision Programming Documentation

Robson Adem

Edge Detection:

I implemented the Sobel edge operator using the method shown below.

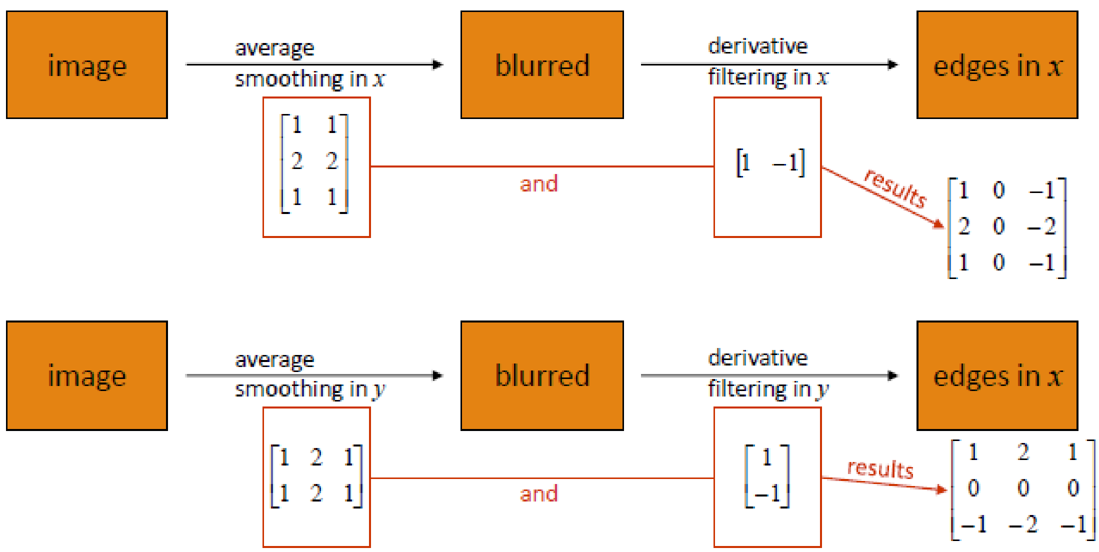


Figure 1: Sobel Edge Detection Algorithm Source Lecture Slides

The combination of the smoothing of the images and the derivative filtering in x yields the Sobel edge operator. As such, I was able to follow the following steps to extract the edges of any given image. The implementation of this algorithm is done with mat lab, and I have listed the respective results and codes for each step.

a) A function that implements the convolution of a 3x3 operator (or kernel) with an image.

```
function [new_img] = convolve(img, kernel)
    % img -image name in the folder e.g 'image2.jpg'
    % kernel- 3x3 matrix kernel
    % By Robson Adem
kernel   = fliplr(flip(kernel,1)); % flipping in the hor and ver direction
img_size = size(img);% This gives the dimensions of the image in [length,width]
length   = img_size(1); %extract the length value from the img_size vector
width    = img_size(2); %extract the width value from the img_size
new_img  = zeros(length,width, 'double');% create a blank matrix for the new img

% The nested for loop computes the convolution
    for i=1:length-2 %edge protection
        for j=1:width-2%edge protection

           % The 9 multiplications needed for the convlution
           new_pixel= kernel(1,1)*img(i,j)+  kernel(1,2)*img(i,j+1) +  kernel(1,3)*img(i,j+2)
           + kernel(2,1)*img(i+1,j)+  kernel(2,2)*img(i+1,j+1) + kernel(2,3)*img(i+1,j+2)
           + kernel(3,1)*img(i+2,j)+  kernel(3,2)*img(i+2,j+1) +  kernel(3,3)*img(i+2,j+2);

           new_img(i+1,j+2)= new_pixel; % put the new pixel in the middle
           % position of the 3x3 kernel overlap with the orginal pixels
           % the edges will be interpolated with 0
        end
    end
end
```

Figure 2: Custom Convolution

The above function flips the kernel and computes a convolution of the kernel with the image in a “3 by 3 window”. For this step, the convolution at the edges is ignored. However, along the edges of the new image from the convolution zeros were assigned as a placeholder. One can decide to crop the edges, but I did not deem it necessary.

b) Using the code from a) convolved the given image (image2.jpg) with the Sobel operator for both x and y direction as indicated in the lecture slide 22.
c) Displayed the output for both x and y component (called it as x_comp.jpg and y_comp.jpg, respectively).
d) Computed the edge response by combining the x and y component. The magnitude at each pixel can be obtained by making use of the equation given in slide 24 of Lecture 4 (Edges).
e) Applied Thresholding to get the final edge map.

```
clear all
img=imread('image2.jpg'); % Read an image
img=im2double(img);
% Matlab realsqrt operator requires double values
% im2double changes 0-255 to double values

sobel_kernel_x=[1,0,-1;
                2,0,-2;
                1,0,-1]; %sobel edge operator

sobel_kernel_y=[1, 2, 1;
                 0, 0, 0;
                -1,-2,-1]; %sobel edge operator

img_size=size(img);% This gives the dimensions of the image in [length,width]
length=img_size(1); %extract the length value from the img_size vector
width=img_size(2); %extract the width value from the img_size

x_comp = convolve(img, sobel_kernel_x); % custom convolution function for kernel 3x3
y_comp = convolve(img, sobel_kernel_y); % custom convolution function for kernel 3x3
img_sobel_edges = realsqrt(x_comp.^2 + y_comp.^2); % sobel edge detection
img_sobel_edges_no_thresholding = img_sobel_edges ;

% Thresholding (YOU CAN CHANGE THE THRESHOLDING VALUE HERE 0.5 seems to show striking result!)
Thresholding=0.5;
for i=1:length
        for j=1:width
            if (img_sobel_edges(i,j)<Thresholding) img_sobel_edges(i,j) = Thresholding;
            end
        end
end
```

Figure 3 : Sobel Edge Detection Implementation by Robson Adem

Following step a), as shown in figure 3, I utilized the function I created to output two new images, one being the derivative in x, and the other being the derivative in y. These two images are displayed in figure 4. Then, I was able to combine these two images to find the edge response of the image. After some meticulous study for a suitable threshold value, I enhanced the initial edge response of the image as shown below in figure 4a and 4b.

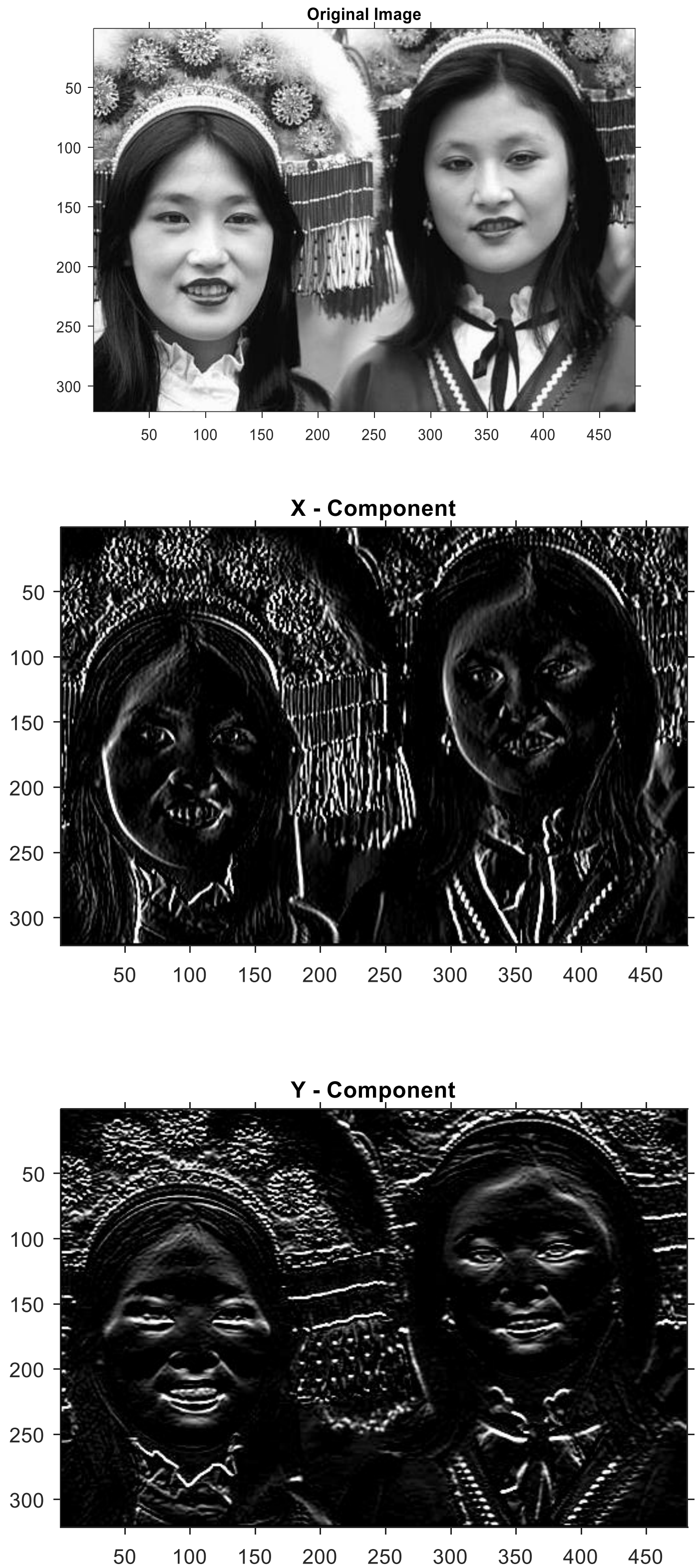


Figure 4a: Original Image and the extracted X and Y components

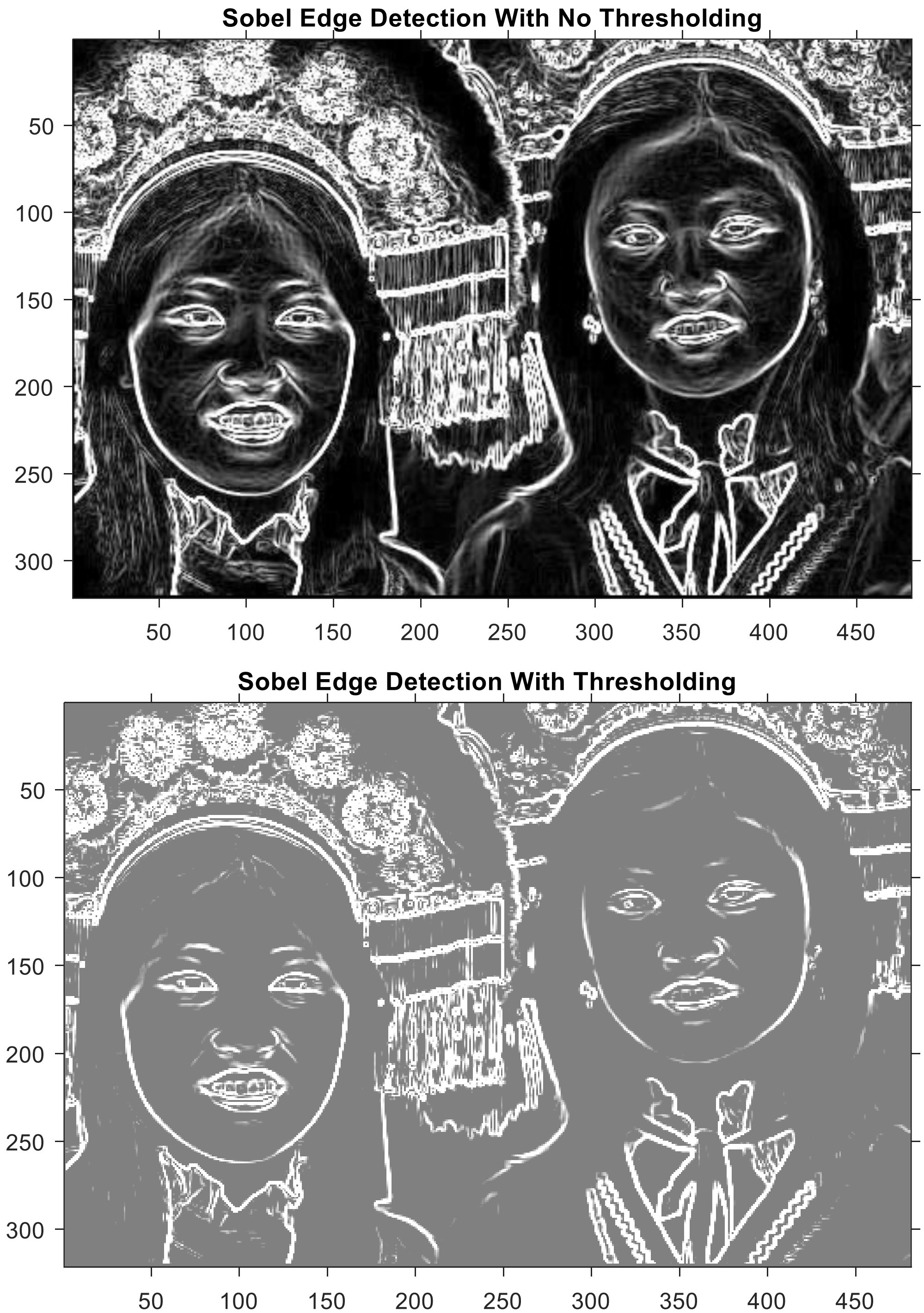


Figure 4b: Sobel Edge Detection with and without thresholding

Corner Detection:

I Implemented the Harris Corner Detector and showcased the outputs with and without thresholding for the two given images.

```
clear all
% img=imread('input_hcd1.jpg'); % Read an image
img=imread('input_hcd2.jpg'); % Read an image
img=im2double(img);

sobel_kernel_x=[1,0,-1;
                2,0,-2;
                1,0,-1]; %sobel edge operatort for blurring

sobel_kernel_y=[1, 2, 1;
                0, 0, 0;
               -1,-2,-1]; %sobel edge operatort for blurring

% Image gradients
img_x=convolve(img, sobel_kernel_x); % custom convolution function for kernel 3x3
img_y=convolve(img, sobel_kernel_y); % custom convolution function for kernel 3x3

% Pixel by Pixel products of the images
Ixx=img_x.*img_x;
Iyy=img_y.*img_y;
Ixy=img_x.*img_y;

window = [1,1,1;
          1,1,1;
          1,1,1]; % smoothing window

img_size=size(img);% This gives the dimensions of the image in [length,width]
length=img_size(1); %extract the length value from the img_size vector
width=img_size(2); %extract the width value from the img_size
R_img = zeros(length,width, 'double');% create a blank matrix for the new img
R_img_no_thresholding = zeros(length,width, 'double');% create a blank matrix for the new img

% Determining R and Thresholding
threshold_value = 15; % 80 for img1 and 15 img2

for i=1:length
        for j=1:width
          Hxy=[ Sxx(i,j) , Sxy(i,j);
                Sxy(i,j),  Syy(i,j)];
          k=0.06;
          R= det(Hxy)- k*trace(Hxy)^2;
          R_img_no_thresholding(i,j) = R;
         if(R > threshold_value) R_img(i,j)= R;
         end

        end
end
```

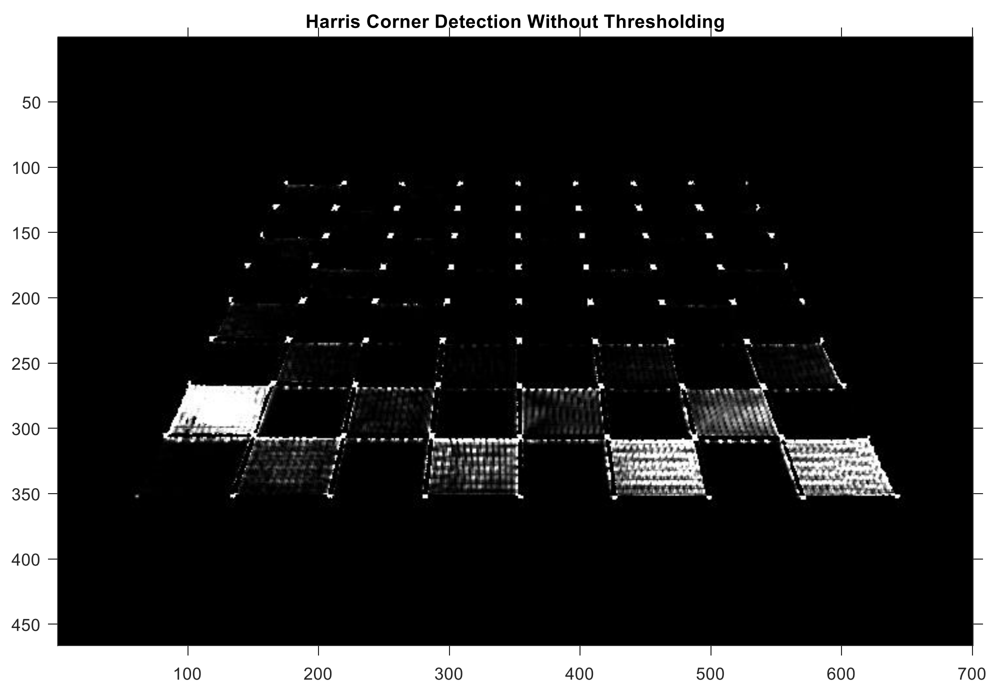
The steps I took to implement the algorithms are as follows.

a) Used the Sobel edge operators in the x and y direction for computing the image derivatives or the gradients Ix and Iy.

b) Found the pixel by pixel products of the gradient images to obtain three new images (Ixx, Iyy, and Ixy).

c) Using the function W(x, y) with a 3x3 kernel of all ones, I obtained the three smoothed images (Sxx, Syy, Sxy).

d) Implemented the measure of cornerness (R) and employed thresholding techniques to find better corners. I have documented the results as follows.



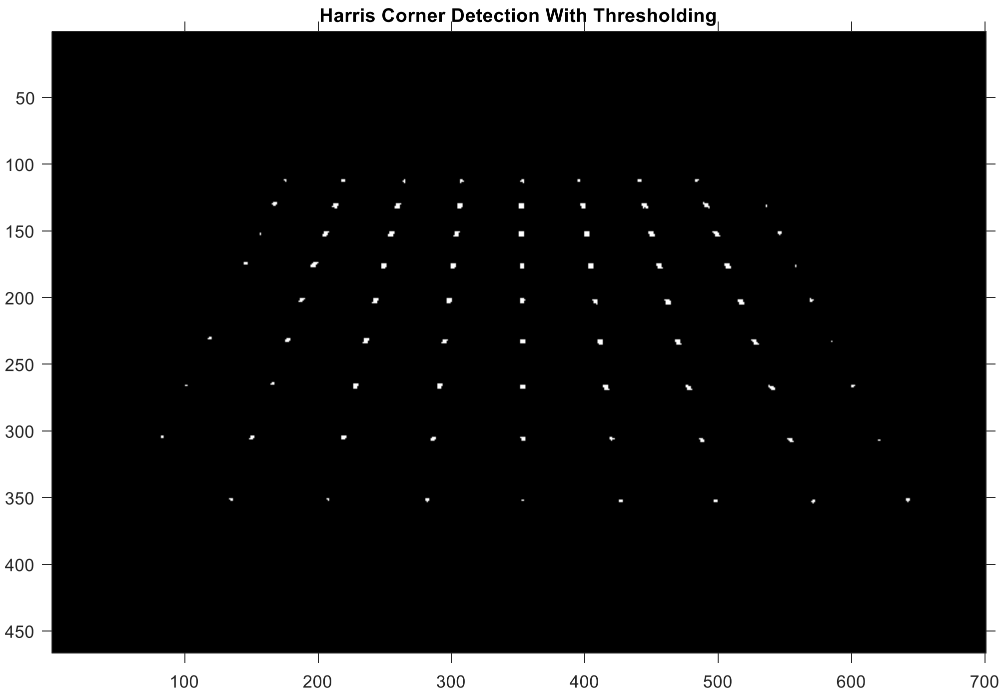
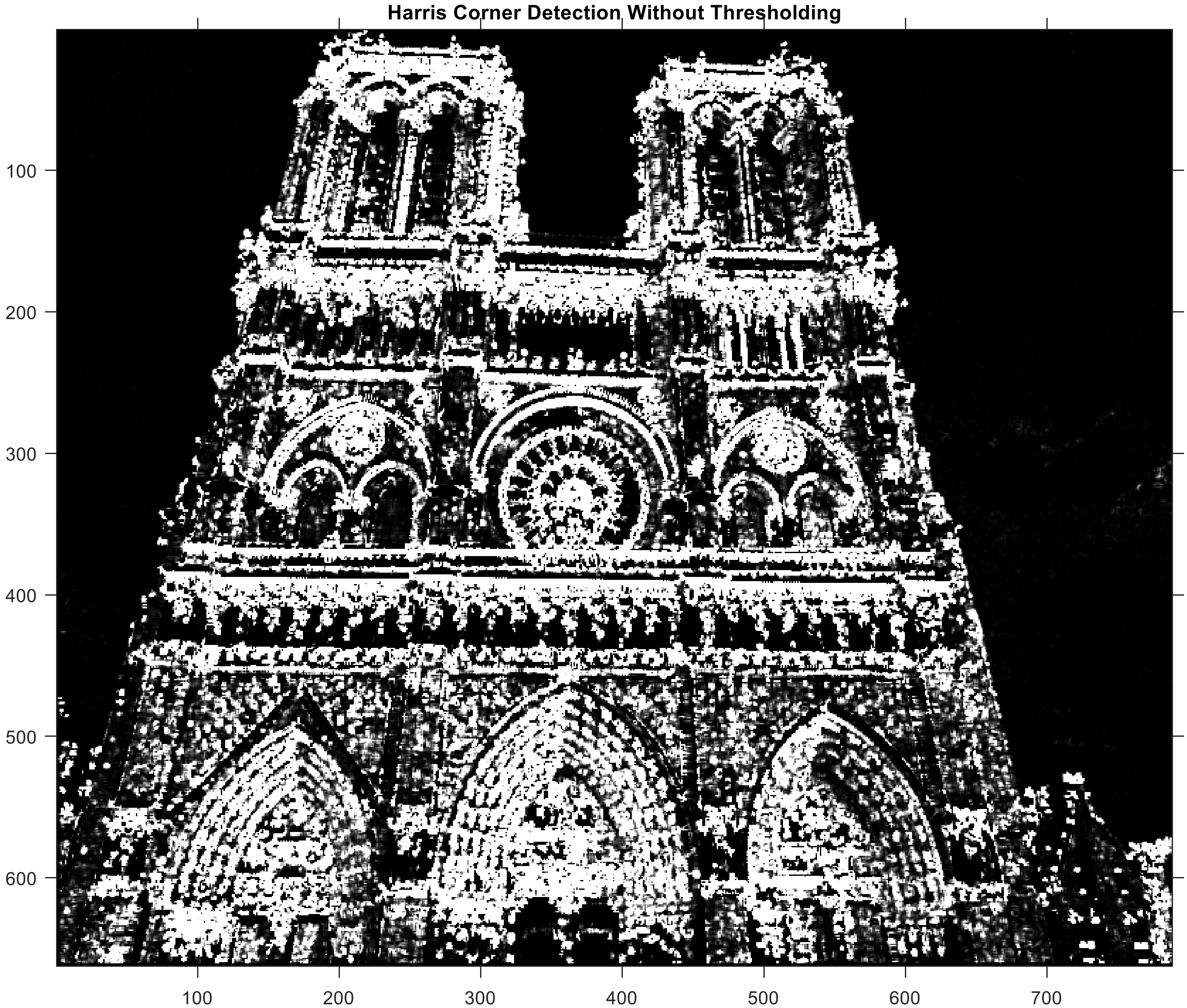


Figure 5: Results for given image 1 (R > 80)


Harris Corner Detection Without Thresholding
100
200
300
400
500
600
100
200
300
400
500
600
700

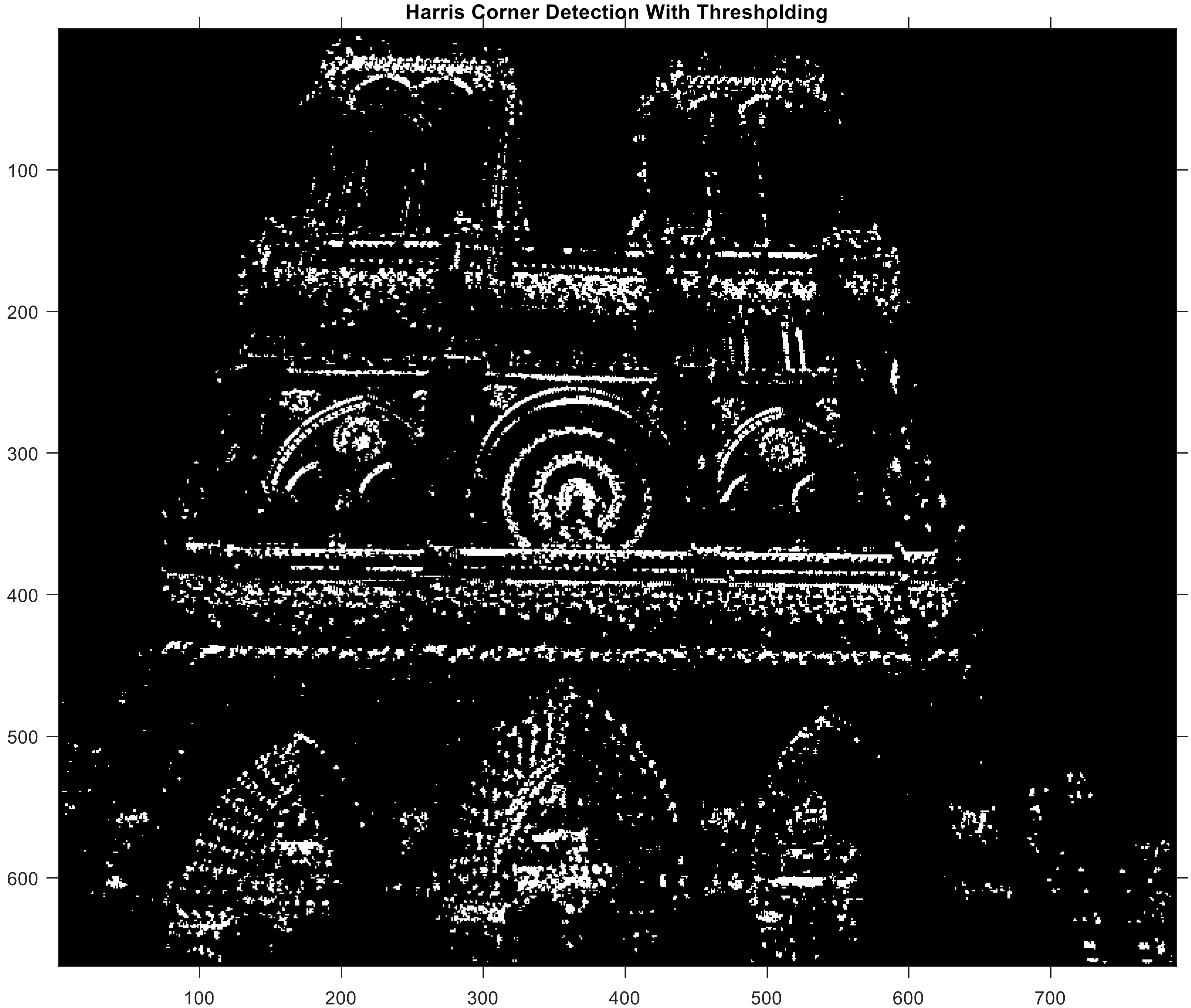


Figure 6: Results for given Image 2 (R > 15)