EE 417 Computer Vision

Berkant Deniz Aktaş

19515

Post-Laboratory Report

October 25, 2018

Sabancı University

Faculty of Engineering and Natural Sciences

Computer Science and Engineering

SOBEL OPERATOR

```
function [img2,img3,img4] = lab3sobel(img)

[row, col, ch] = size(img);
if(ch==3)
   img = rgb2gray(img);
end

img = double(img);
K=1;
x = [ -1 0 1 ; -2 0 2; -1 0 1 ];
y = [ 1 2 1 ; 0 0 0; -1 -2 -1 ];
for i=K+1:1:row-K-1
   for j=K+1:1:col-K-1
      subImg = img(i-K:i+K,j-K:j+K);
      valueX = sum(sum(subImg.*x));
      valueY = sum(sum(subImg.*y));

      img2(i,j)= valueX;
      img3(i,j)= valueY;
      valueGrad = sqrt(valueX^2 + valueY^2);
      img4(i,j) = valueGrad;
      if(valueGrad > 100)
         img5(i,j)=valueGrad;
      end
   end
end

img = uint8(img);
img2 = uint8(img2);
img3 = uint8(img3);
img4 = uint8(img4);
figure;
subplot(1,5,1);
imshow(img);
title('Original Image')
subplot(1,5,2);
imshow(img3);
title('Sobel Horizontal Image')
subplot(1,5,3);
imshow(img2)
title('Sobel Vertical Image')
subplot(1,5,4);
imshow(img4);
title('Sobel Gradient Image')
subplot(1,5,5);
imshow(img5);
title('Sobel Edges Image')
end
```

We implemented horizontal and vertical Sobel operators in previous labs. They were detecting the horizontal and vertical edges separately. In this lab, we learned to combine two values and combine horizontal and vertical edges. Unfortunately, it was not sufficient for

proper edge detecting. After adding a threshold value, we saw thinner lines. As one can guess increasing the threshold value will cause decrease of the number of the edges detected. Only strong edges which has high derivate value will be displayed on the processed image.

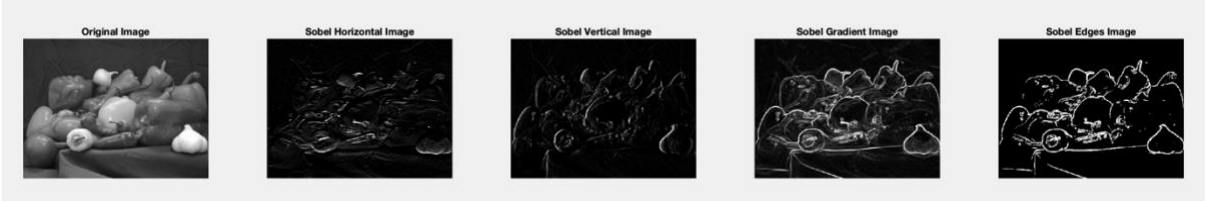

Figure 1. Sobel on peppers.png

PREWITT OPERATOR

```
function [img2,img3,img4] = lab3prewitt(img)

[row, col, ch] = size(img);
if(ch==3)
   img = rgb2gray(img);
end

img = double(img);
K=1;
x = [ -1 0 1 ; -1 0 1; -1 0 1 ]; % for vertical edges
y = [ -1 -1 -1 ; 0 0 0; 1 1 1 ]; % for horizontal edges
for i=K+1:1:row-K-1
   for j=K+1:1:col-K-1

      subImg = img(i-K:i+K,j-K:j+K);
      valueX = sum(sum(subImg.*x));
      valueY = sum(sum(subImg.*y));
      img2(i,j)= valueX;
      img3(i,j)= valueY;
      valueGrad = sqrt(valueX^2 + valueY^2);
      img4(i,j) = valueGrad;
      if(valueGrad > 100)
         img5(i,j)=valueGrad;
      end
   end
end

img = uint8(img);
img2 = uint8(img2);
img3 = uint8(img3);
img4 = uint8(img4);
figure;
subplot(1,5,1);
imshow(img);
title('Original Image')
subplot(1,5,2);
imshow(img3);
title('Prewitt Horizontal Image')
```

```
subplot(1,5,3);
imshow(img2)
title('Prewitt Vertical Image')
subplot(1,5,4);
imshow(img4);
title('Prewitt Gradient Image')
subplot(1,5,5);
imshow(img5);
title('Prewitt Edges Image')
end
```

Similar code is used for Prewitt edge detector. I only changed vertical and horizontal masks and achieved a similar result.

Figure 2. Prewitt on peppers.png

SOBEL VS PREWITT

Both methods are using similar technique, using a mask and iterating over picture.

Middle rows are empty, so mask can find the derivative between pixels in columns or rows and detect edges. The difference is the mask values. Same mask is used except the middle value of the column or row. Prewitt has a fixed value 1 but Sobel's value is not fixed, and weight of the mask can be increased. Most of the cases, like in our lab, the weight is 2 for Sobel. Increasing the weight will cause more difference between pixels and we will have more edges detected.

LAPLACIAN OF GAUSSIAN

```
function img = lab3log(imgOrg)

[row, col, ch] = size(imgOrg);
if(ch==3)
   imgOrg = rgb2gray(imgOrg);
end

imgOrg = double(imgOrg);
img = zeros(size(imgOrg));
img2 = zeros(size(imgOrg));

K=2;
k = (1/273)*[1 4 7 4 1; 4 16 26 16 4; 7 26 41 26 7; 4 16 26 16 4; 1 4 7 4 1 ];

for i=K+1:1:row-K-1
   for j=K+1:1:col-K-1
      subImg = imgOrg(i-K:i+K,j-K:j+K);
      value = sum(sum(subImg.*k));
      img(i,j)= value;
   end
end

K=1;
l = [0 -1 0; -1 4 -1; 0 -1 0];
for i=K+1:1:row-K-1
   for j=K+1:1:col-K-1
      subImg = img(i-K:i+K,j-K:j+K);
      value2 = sum(sum(subImg.*l));
      img2(i,j)= value2;
   end
end

imgOrg = uint8(imgOrg);


figure;
subplot(1,3,1);
imshow(imgOrg);
subplot(1,3,2);
imshow(img2,[]);
subplot(1,3,3);
plot(img2(20:1:60,100));
end
```

The idea of LoG is different than previous edge detectors. Prewitt and Sobel takes the first order derivative but LoG works on second order derivative. Unlike others this filter could be formed with [0 -1 0; -1 4 -1; 0 -1 0] or [-1 -1 -1; -1 8 -1; -1 -1 -1]. Both of masks are taking

the second order derivative. As it is stated in the lecture taking the derivative of the image is risky if it is noisy. Noise will increase with derivatives so idea of LoG comes from here. Smoothing the image with Gaussian Smoothing before applying the Laplacian Operator is the basis of LoG. One can convolve the Gaussian Operator and Laplacian Operator and achieve another mask for efficiency but in this lab I Iterated the image two times and applied Gaussian mask then Laplacian mask.

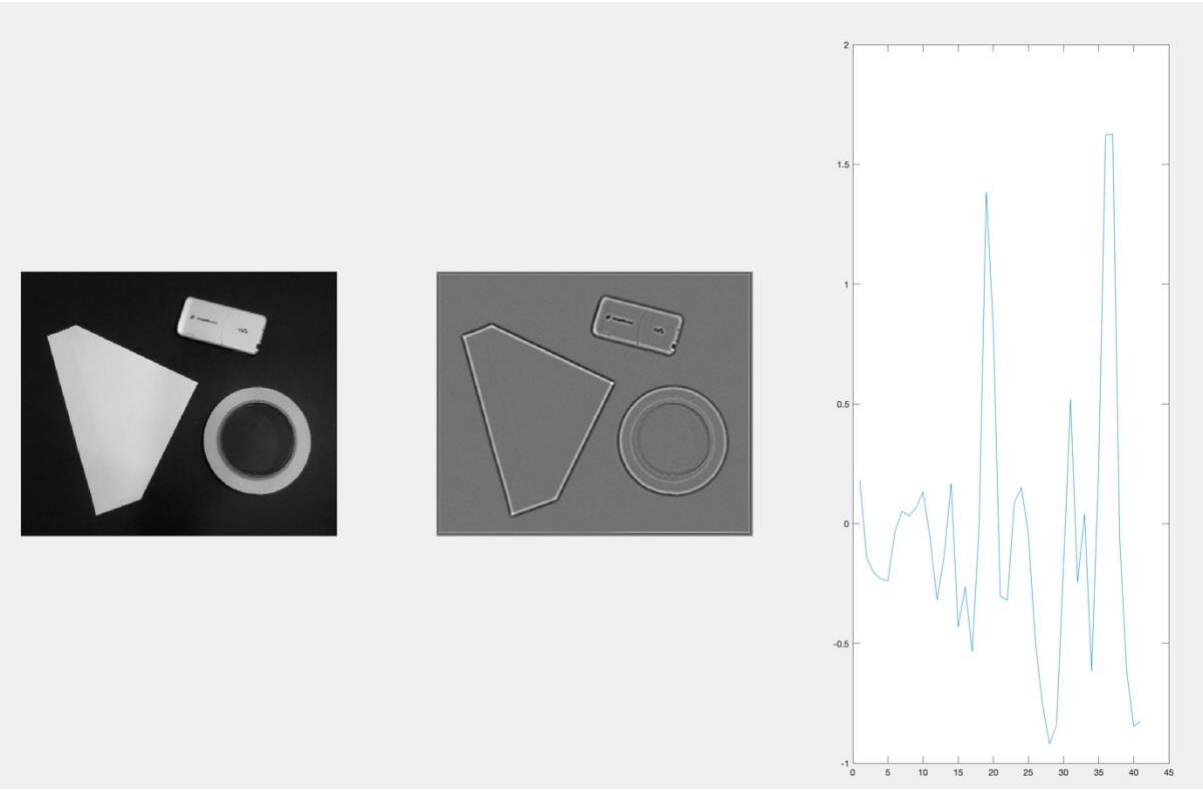

Figure 3. LoG on Object_contours.jpg

KANADE-TOMASI

We used Kanade-Tomasi algorithm in order to detect the features (corners) in the image. The algorithm has been improved since its foundation. We used a version which finds the

gradients of the sub image and computes the Hessian matrix of the window in order to find eigen values. If the minimum eigan value is greater than our threshold we can classify it as a corner or a feature of the image. Note that increasing the thresh hold will decrease the number of corners detected. Also, for loop iteration is important for detection of the corners. Incrementing i and j one by one will cause lots of corners detected because it will try to find an edge every window. I used 10 as an incrementation number.

```
function img2= lab3ktcorners(img)
[row, col, ch] = size(img);
if(ch==3)
   img = rgb2gray(img);
end
img = double(img);
img2 = zeros(size(img));

[Gx,Gy] =  imgradientxy(img);
K=1;

corners = [];
for i=K+1:10:row-K-1
   for j=K+1:10:col-K-1

Gsubx = Gx(i-K:i+K,j-K:j+K);
Gsuby = Gy(i-K:i+K,j-K:j+K);
       a=sum(sum(Gsubx.*Gsubx));
       b=sum(sum(Gsubx.*Gsuby));
       c=sum(sum(Gsuby.*Gsuby));

        H = [ a b ; b c ];

        eigs = eig(H);
        if(min(eigs) > 200 )
          corners = [corners; i j];
        end
   end
end

    img = uint8(img);
       figure;
       imshow(img);
       hold on;
       plot(corners(:,2),corners(:,1),'r*');
end
```

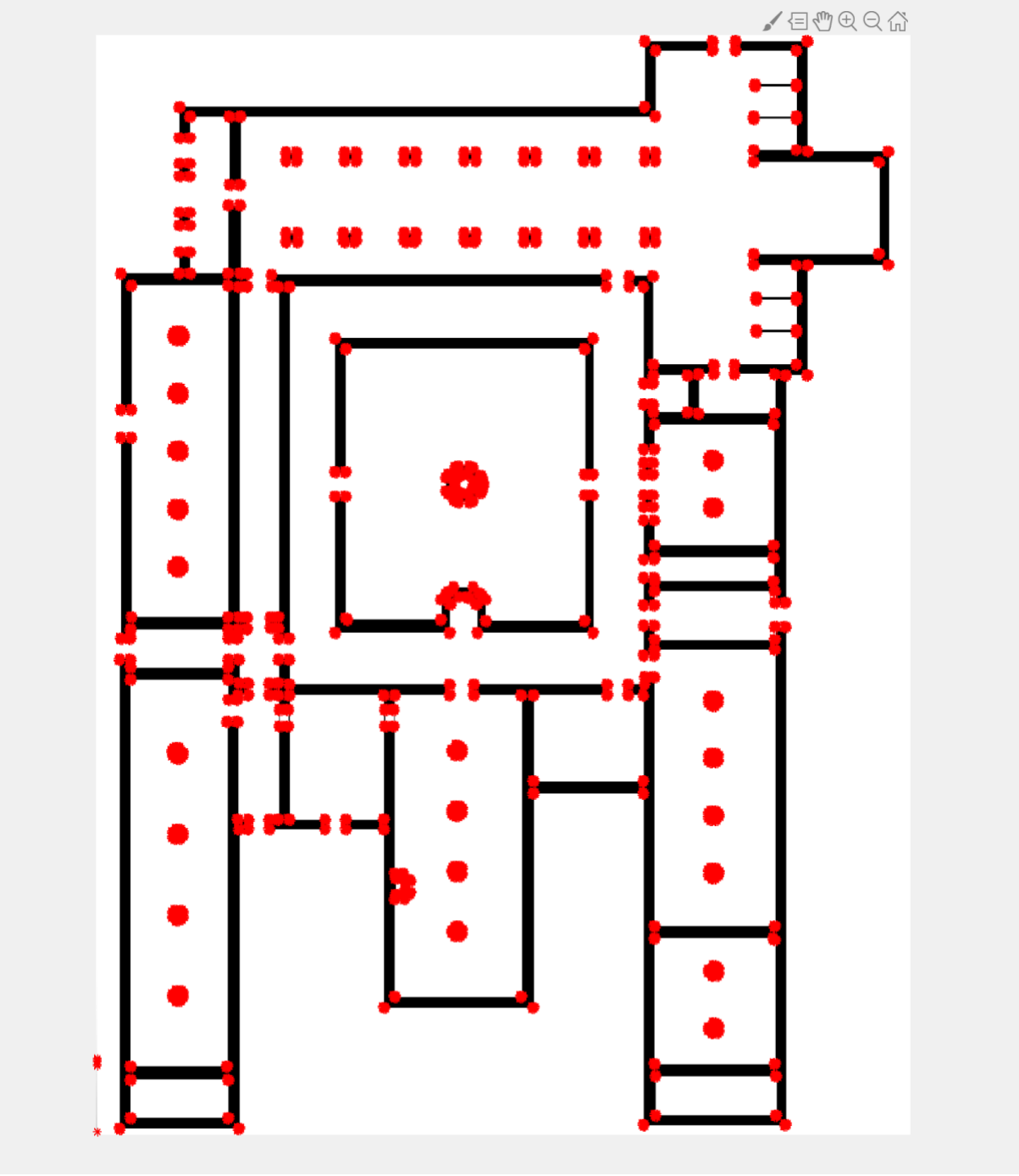

Figure 4. KT on Monastry.bmp with 1 incrementation

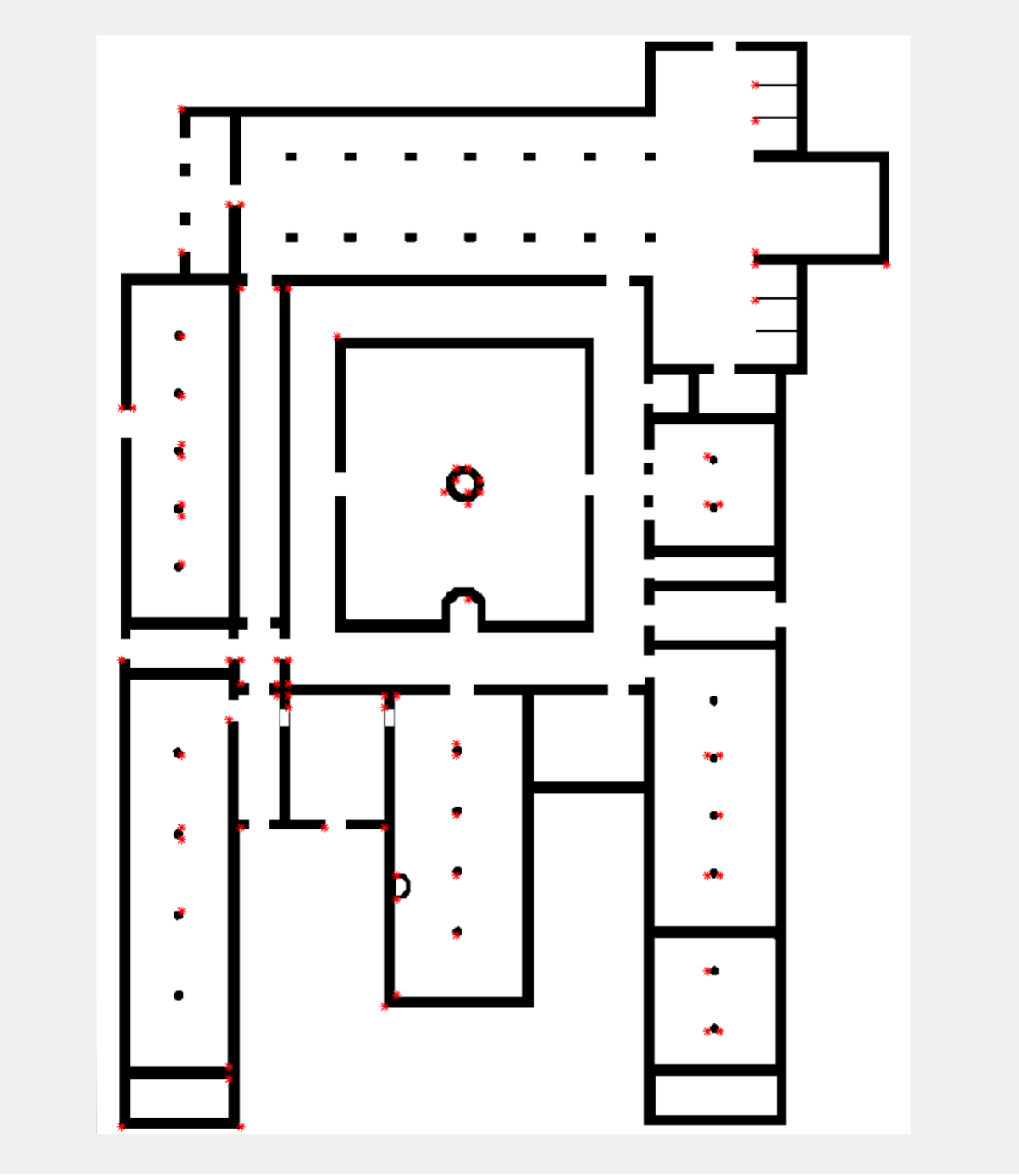

Figure 5. KT on Monastry.bmp with 10 incrementation

As expected, incrementing the incrementation of the double for loop resulted less corners detected. Figure 6. Found all the corners in the image.

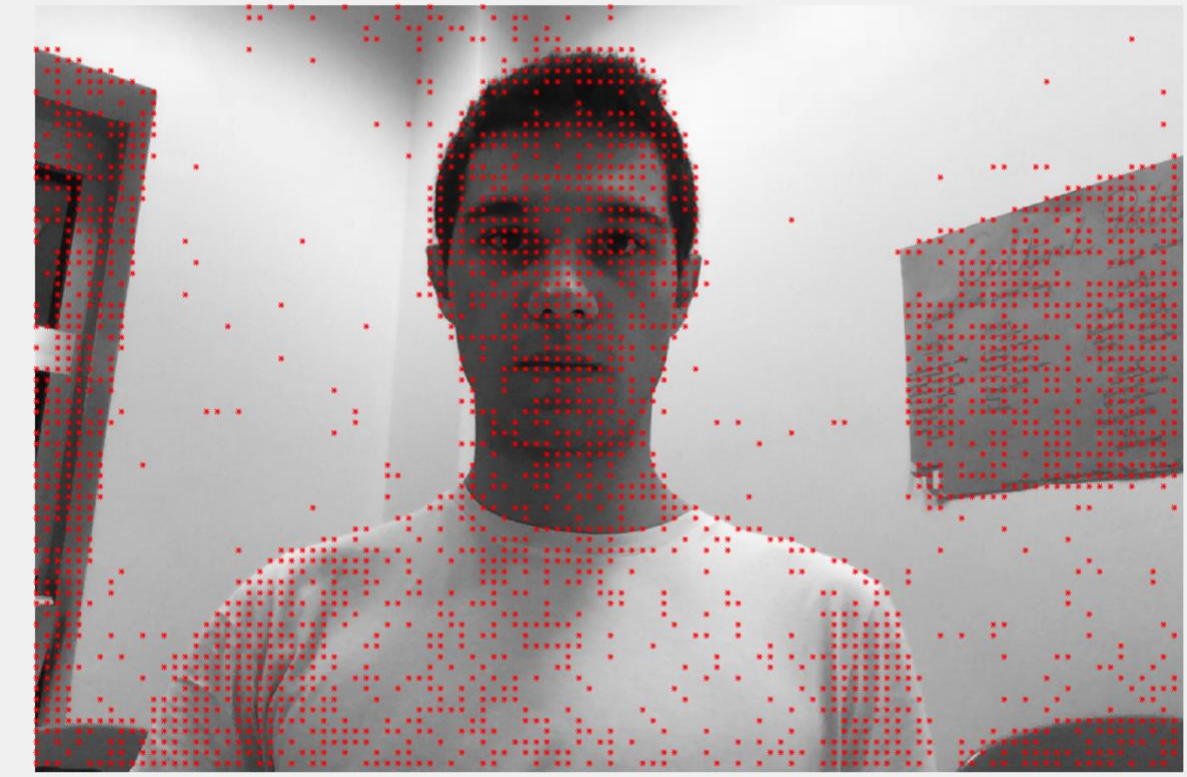

Figure 6. KT on Monastry.bmp with 10 incrementation and threshold 100.

Figure 7. KT on Monastry.bmp with 10 incrementation and threshold 800

Comparison of the Figure 7. and Figure 8. will show importance of the threshold value. Since I incremented the threshold, less corners detected. The image is noisy so the results are not robust.