Corner Detection (Jianda Wang)

1) First, apply Gaussian smoothing (with the standard deviation) to an input image I, to obtain Is.

Code for Gaussian smoothing:

```
I1 = imread('CheckerBoard.jpg');
I1 = rgb2gray(I1);

k = 1;   %deviation

n = 5*k;

I1 = double(I1);
n1 = floor((n+1)/2);
[Iwidth,Ilength] = size(I1);

for i = 1:n
      a(i) = exp((-(i-n1)^2)/(2*k^2))/(k*sqrt(2*pi));
end

a = a/a(1);
a = floor(a);
b = a/sum(a);

C(1:Iwidth,1:Ilength) = 1;

 for i = 1:Iwidth
   C1 = conv2(I1(i,:),b,'same');
   C(i,:) = C1;
 end

b1 = b';

  for i = 1:Ilength
   C2 = conv2(C(:,i),b1,'same');
   C(:,i) = C2;
  end

Is = C;
Is = uint8(Is);
```

2) Implement the corner detection algorithm (CORNERS), by using Is as input, as describe in class and also in the textbook.

```
N = 5;
low = 3500;


Jx(1:Iwidth,1:Ilength) = 1;
Jy(1:Iwidth,1:Ilength) = 1;

a = [1 0 -1];

for i = 1:Iwidth
   C1 = conv2(single(Is(i,:)),single(a),'same');
   Jx(i,:) = C1;
end

a = [-1;0;1];

for i = 1:Ilength
   C2 = conv2(single(Is(:,i)),single(a),'same');
   Jy(:,i) = C2;
end

l = Iwidth*Ilength;
Lx(1:l) = 0;
Ly(1:l) = 0;
Lm(1:l) = 0;
l1 = 1;

for i = N+1:Iwidth-N
   for j = N+1:Ilength-N

      a = 0;
      b = 0;
      c = 0;

      for i1 = i-N:i+N
         for j1 = j-N:j+N
```

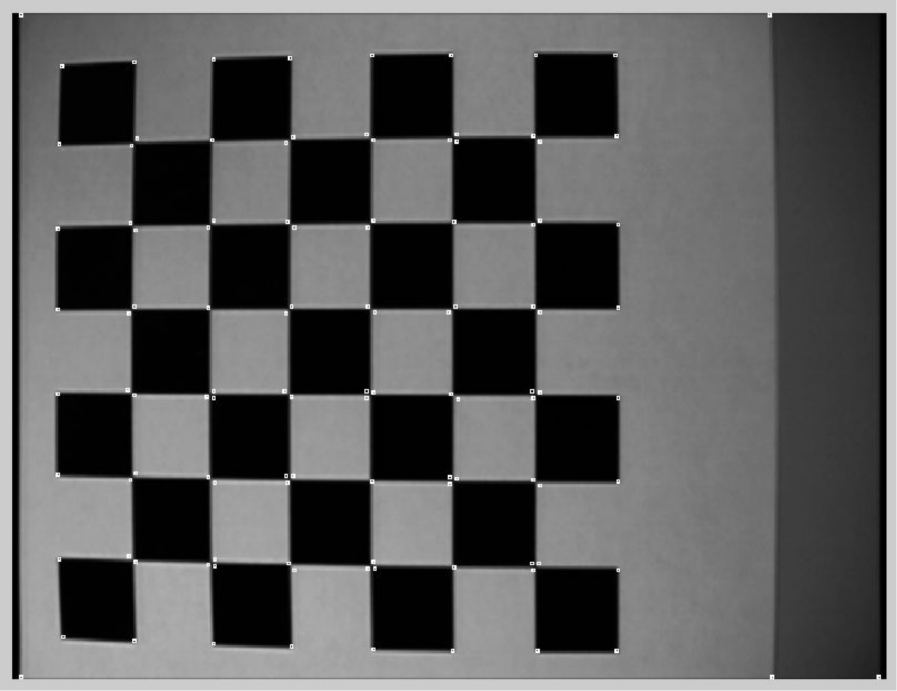
```
                a = a+Jx(i1,j1)*Jx(i1,j1);
                b = b+Jx(i1,j1)*Jy(i1,j1);
                c = c+Jy(i1,j1)*Jy(i1,j1);
            end
        end

        D = [a b;b c];
        e = eig(D);
        m = min(e);

        if m>low
            Lx(l1) = i;
            Ly(l1) = j;
            Lm(l1) = m;
            l1 = l1+1;
        else
            l1 = l1;
        end
        end
end

Lx(Lx==0) = [];
Ly(Ly==0) = [];
Lm(Lm==0) = [];

[Lm,ind] = sort(Lm,'descend');

n = length(Lm);
Mx(1:n) = 0;
My(1:n) = 0;

for i = 1:n
    Mx(i) = Lx(ind(i));
    My(i) = Ly(ind(i));
end

Lx = Mx;
Ly = My;
i1 = 1;

for i = i1:n-1
    for j = i+1:n
        if
Lx(j)>=Lx(i)-2*N-2&&Lx(j)<=Lx(i)+2*N+2&&Ly(j)>=Ly(i)-2*N-2&&Ly(j)<=Ly
```

```
(i)+2*N+2
            Lx(j) = 0;
            Ly(j) = 0;
        else
            Lx(j) = Lx(j);
            Ly(j) = Ly(j);
        end
    end
end

Lx(Lx==0) = [];
Ly(Ly==0) = [];

n = length(Lx);

for i = 1:n
    a = Lx(i)-N;
    b = Ly(i)-N;

    for j = 0:2*N
        I1(a,b+j) = 255;
        I1(a+2*N,b+j) =255;
    end

    for j = 1:2*N-1
        I1(a+j,b) = 255;
        I1(a+j,b+2*N) = 255;
    end
end

I1 = uint8(I1);
figure;
imshow(I1);
```

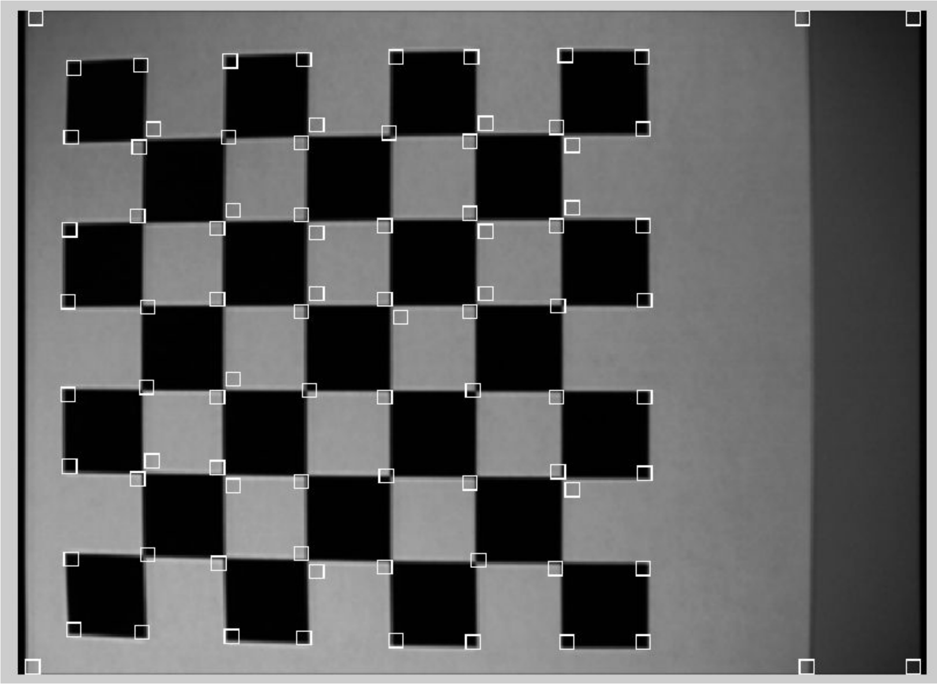
3) Test your corner algorithm on images “Building1.jpg” and “CheckerBoard.jpg”. Try different values of the σ , the neighborhood size, and the threshold (τ) on λ_2 . Compare and evaluate your results.

“Checkerboard.jpg”:

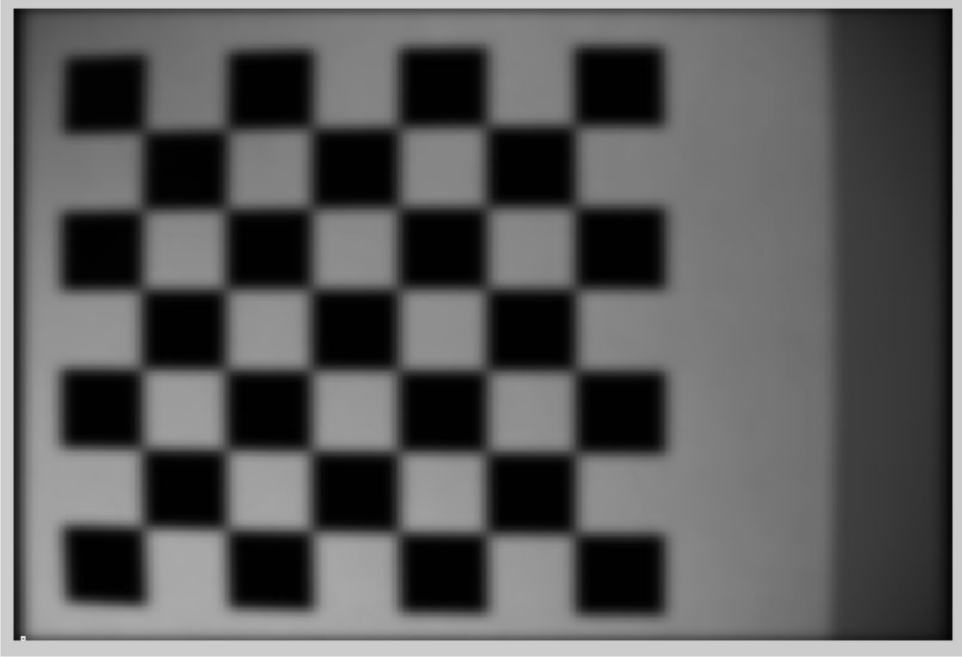
(a) With σ =1, n=3, threshold is 700



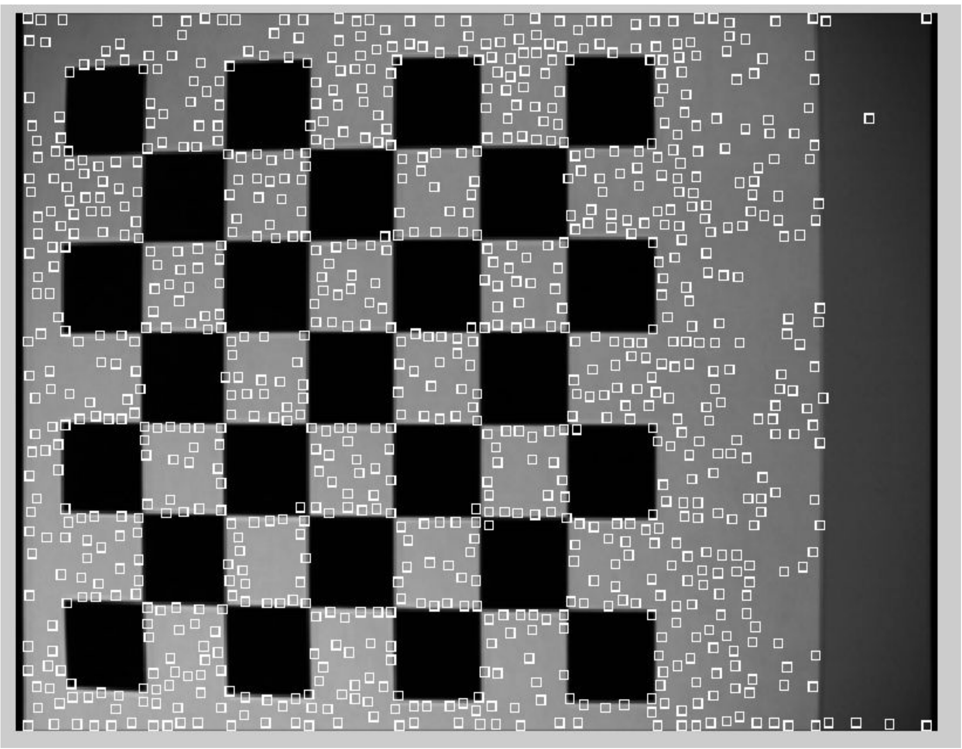
(b) With σ =1, n=11, threshold is 3500



(c) With $\sigma = 5$, n=3, threshold is 700

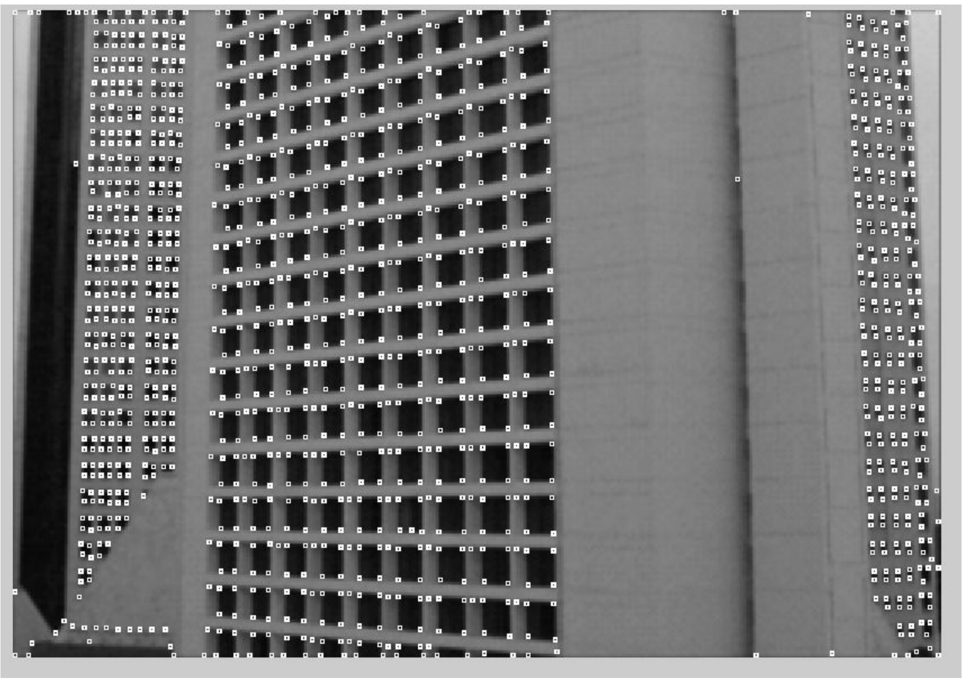


(d) With $\sigma = 1$, n=7, threshold is 100

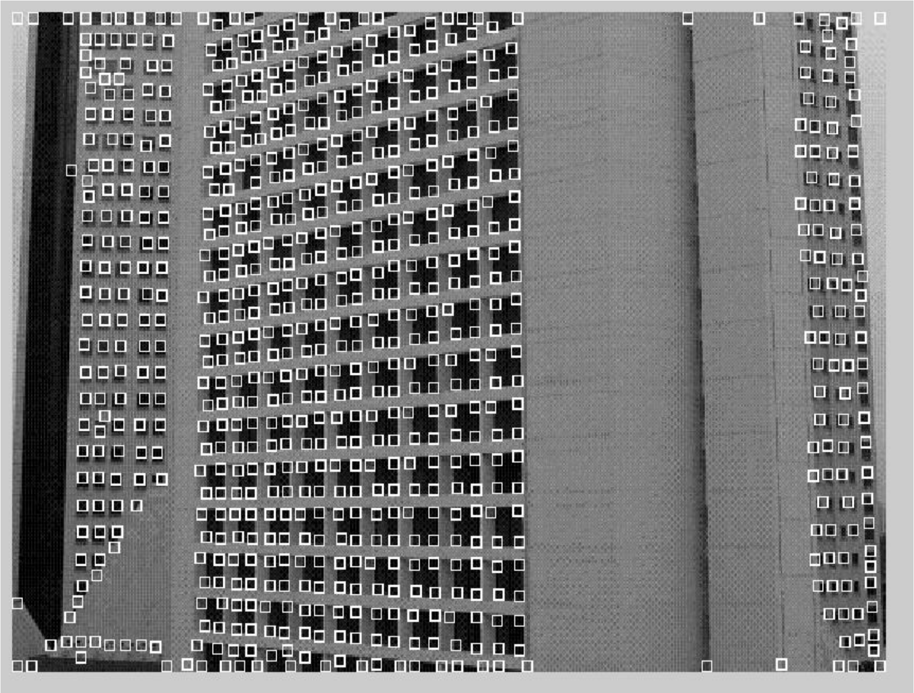


“Building1.jpg”

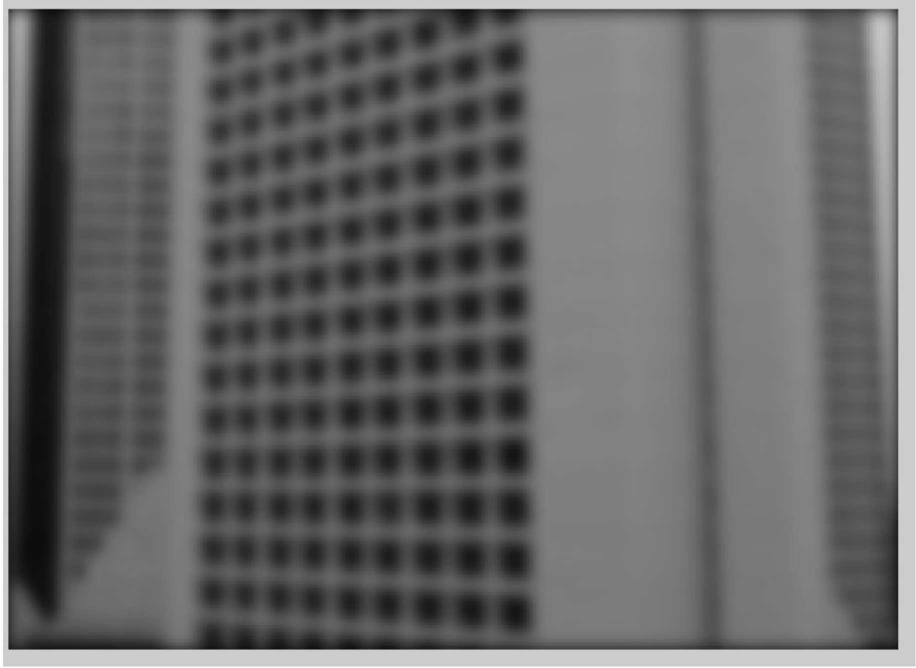
(a) With σ =1, n=3, threshold is 700



(b) With σ =1, n=7, threshold is 2100



(c) With σ =5, n=3, threshold is 700



Conclusion:

(1) When we increase the size of neighborhood, we should increase the threshold at the same time. Otherwise, we will get some squares that appear on the places where are not the corners.

This is because the eigenvalues will become larger if we increase the size of the neighborhood. It works well when we choose τ = 300n-300;

(2) When the deviation increases, we get less corners on the image.

This is because Gaussian smoothing will smooth the image so that the eigenvalues become smaller. If we do not decrease the threshold, some corners will be lost.

(3) In the fourth step of the algorithm Corners, we should delete all the points appear on the list which belong to TWO SIZE of the neighborhood of p so that the squares will not overlap.

Code for test:

```
function I = JiandaCorner(Image,k,n1,low)
%Image is the array of an image; k is deviation; n1 is the size of 
neighnorhood: low is threshold;

I1 = Image;

N = (n1-1)/2;

n = 5*k;

I1 = double(I1);
n1 = floor((n+1)/2);
[Iwidth,Ilength] = size(I1);

for i = 1:n
      a(i) = exp((-(i-n1)^2)/(2*k^2))/(k*sqrt(2*pi));
end

a = a/a(1);
a = floor(a);
b = a/sum(a);

C(1:Iwidth,1:Ilength) = 1;

 for i = 1:Iwidth
   C1 = conv2(I1(i,:),b,'same');
   C(i,:) = C1;
 end

b1 = b';

  for i = 1:Ilength
   C2 = conv2(C(:,i),b1,'same');
   C(:,i) = C2;
  end

Is = C;
Is = uint8(Is);

Jx(1:Iwidth,1:Ilength) = 1;
Jy(1:Iwidth,1:Ilength) = 1;
```

```
a = [1 0 -1];

for i = 1:Iwidth
   C1 = conv2(single(Is(i,:)),single(a),'same');
   Jx(i,:) = C1;
end

a = [-1;0;1];

for i = 1:Ilength
   C2 = conv2(single(Is(:,i)),single(a),'same');
   Jy(:,i) = C2;
end

l = Iwidth*Ilength;
Lx(1:l) = 0;
Ly(1:l) = 0;
Lm(1:l) = 0;
l1 = 1;

for i = N+1:Iwidth-N
   for j = N+1:Ilength-N

      a = 0;
      b = 0;
      c = 0;

      for i1 = i-N:i+N
         for j1 = j-N:j+N
            a = a+Jx(i1,j1)*Jx(i1,j1);
            b = b+Jx(i1,j1)*Jy(i1,j1);
            c = c+Jy(i1,j1)*Jy(i1,j1);
         end
      end

      D = [a b;b c];
      e = eig(D);
      m = min(e);

      if m>low
         Lx(l1) = i;
         Ly(l1) = j;
         Lm(l1) = m;
         l1 = l1+1;
```

```
        else
            l1 = l1;
        end
        end
end

Lx(Lx==0) = [];
Ly(Ly==0) = [];
Lm(Lm==0) = [];

[Lm,ind] = sort(Lm,'descend');

n = length(Lm);
Mx(1:n) = 0;
My(1:n) = 0;

for i = 1:n
    Mx(i) = Lx(ind(i));
    My(i) = Ly(ind(i));
end

Lx = Mx;
Ly = My;
i1 = 1;

for i = i1:n-1
    for j = i+1:n
        if
Lx(j)>=Lx(i)-2*N-2&&Lx(j)<=Lx(i)+2*N+2&&Ly(j)>=Ly(i)-2*N-2&&Ly(j)<=Ly
(i)+2*N+2
            Lx(j) = 0;
            Ly(j) = 0;
        else
            Lx(j) = Lx(j);
            Ly(j) = Ly(j);
        end
    end
end

Lx(Lx==0) = [];
Ly(Ly==0) = [];

n = length(Lx);
```

```
for i = 1:n
   a = Lx(i)-N;
   b = Ly(i)-N;

   for j = 0:2*N
      Is(a,b+j) = 255;
      Is(a+2*N,b+j) =255;
   end

   for j = 1:2*N-1
      Is(a+j,b) = 255;
      Is(a+j,b+2*N) = 255;
   end
end

Is = uint8(Is);
I = Is;
end
```