Homework 3 - SOLUTIONS

Due Monday, February 18, 2013 (by 8:00 pm)

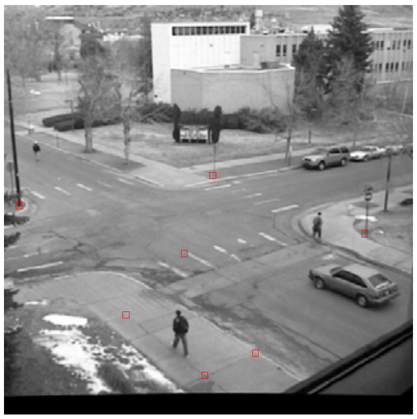
Notes: Please email me your solutions for these problems (in order) as a single Word or PDF document. If you do a problem on paper by hand, please scan it in and paste it into the document (although I would prefer it typed!).

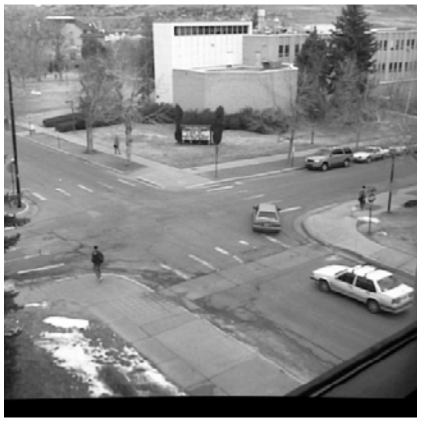
1. **(15 pts) On the course website are two images of a street intersection taken from a stationary camera, taken 5 seconds apart. Transform these images to "orthophotos"; *i.e.*, images taken from a viewpoint directly overhead, such that the scale is uniform. Here are some control points that have been measured in the scene:**

Image (x,y)	Actual (x,y) in feet
154, 389	0, 0
453, 287	53.5, -1
263, 214	60.5, -54.5
20, 252	5, -56
253, 464	0, 17.7
316, 436	8, 17.7
227, 312	19.9, -14.6

Choose a scale of one pixel = 0.25 feet in the output image. How fast (miles per hour) is the person in the lower left walking (it's ok to measure the location of the person in the orthophotos by hand)?

Solution: The two images:



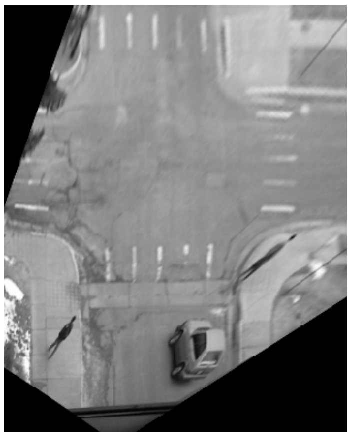


The Matlab code:

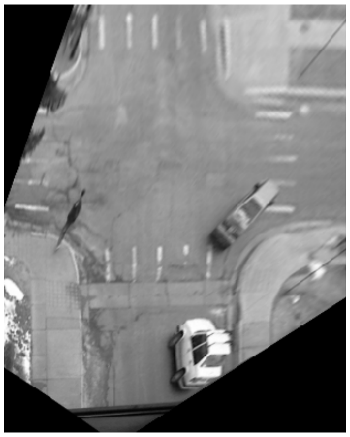
```
% HW3 p1
clear all
close all
```

```
I1 = imread('image0037.jpg');
imshow(I1, []);

% Image points (x,y)
p1 = [
    154 389;    % Origin
    453 287;    % Stop
    263 214;    % Stop
    20  252;    % Lamp
    253 464;    % Sidewalk 1
    316 436;    % Sidewalk 2
    227 312];   % White mark

% Draw marks
for i=1:size(p1,1)
    x = p1(i,1);    y = p1(i,2);
    rectangle('Position', [x-4 y-4 8 8], 'EdgeColor', 'r');
end

% Corresponding world coordinates (x,y in feet)
p2 = [
    0   0;
    53.5    -1;
    60.5    -54.5;
    5   -56;
    0   17.7;
    8   17.7;
    19.9    -14.6];
% Scale so that one pixel = S feet
S = 0.25;
p2 = p2/S;

T = cp2tform(p1,p2, 'projective');
I1ortho = imtransform(I1, T, 'XData', [-10 70]/S, 'YData', [-70 30]/S);
figure, imshow(I1ortho, []);
impixelinfo
% Position of person's feet = (43, 327) measured by hand

I2 = imread('image0187.jpg');
figure, imshow(I2, []);
I2ortho = imtransform(I2, T, 'XData', [-10 70]/S, 'YData', [-70 30]/S);
figure, imshow(I2ortho, []);
impixelinfo
% Position of person's feet = (50, 230) measured by hand

d = S*sqrt((50-43)^2 + (327-230)^2);
fprintf('Person traveled %f feet in 5 seconds, or %f mph\n', ...
    d, (d/5)*0.68);
```

The orthophotos:



Pixel info: (24, 152) 140



Pixel info: (20, 151) 136

Person traveled 24.313062 feet in 5 seconds, or 3.306576 mph

2. (15 pts) The auto-correlation score is $E_{AC}(\Delta\mathbf{u}) = \sum_i w(\mathbf{x}_i)[\nabla I_0(\mathbf{x}_i)\cdot\Delta\mathbf{u}]^2$ (equation 4.5 in the textbook). Show that this equals $\Delta\mathbf{u}^T\mathbf{A}\Delta\mathbf{u}$, where matrix A is as given in equation 4.8.

Solution:

$$
\begin{aligned}
E_{AC}(\Delta\mathbf{u}) &= \sum_i w(\mathbf{x}_i)[\nabla I_0(\mathbf{x}_i)\cdot\Delta\mathbf{u}]^2 \\
&= \sum_i w(\mathbf{x}_i)\left[\Delta x\frac{\partial I_0(\mathbf{x}_i)}{\partial x} + \Delta y\frac{\partial I_0(\mathbf{x}_i)}{\partial y}\right]^2 \\
&= \sum_i w(\mathbf{x}_i)\left[\Delta x\left(\frac{\partial I_0(\mathbf{x}_i)}{\partial x}\right)^2\Delta x + 2\Delta x\left(\frac{\partial I_0(\mathbf{x}_i)}{\partial x}\right)\left(\frac{\partial I_0(\mathbf{x}_i)}{\partial y}\right)\Delta y + \Delta y\left(\frac{\partial I_0(\mathbf{x}_i)}{\partial y}\right)^2\Delta y\right] \\
&= \Delta x\left[\sum_i w(\mathbf{x}_i)\left(\frac{\partial I_0(\mathbf{x}_i)}{\partial x}\right)^2\right]\Delta x + 2\Delta x\left[\sum_i w(\mathbf{x}_i)\left(\frac{\partial I_0(\mathbf{x}_i)}{\partial x}\right)\left(\frac{\partial I_0(\mathbf{x}_i)}{\partial y}\right)\right]\Delta y + \Delta y\left[\sum_i w(\mathbf{x}_i)\left(\frac{\partial I_0(\mathbf{x}_i)}{\partial y}\right)^2\right]\Delta y \\
&= \Delta x\left(w * I_{xx}\right)\Delta x + 2\Delta x\left(w * I_{xy}\right)\Delta y + \Delta y\left(w * I_{yy}\right)\Delta y \\
&= \begin{pmatrix}\Delta x & \Delta y\end{pmatrix}\begin{pmatrix} w * I_{xx} & w * I_{xy} \\ w * I_{xy} & w * I_{yy}\end{pmatrix}\begin{pmatrix}\Delta x \\ \Delta y\end{pmatrix}
\end{aligned}
$$

3. (20 pts) Take the Matlab corner detector program developed in class and make the following changes:

- **a. Instead of using a square region of size NxN to sum the gradient products, weight the values, using a Gaussian mask for w(x,y).**
- **b. Instead of using the interest point measure det(A)/trace(A), use the minimum eigenvalue of A[1]. This is the "Shi-Tomasi" approach.**
- **c. Instead of taking all interest points above a minimum threshold, take the 10 points with the highest scores.**

Apply this program to find the top 10 corner points in the image "test000.jpg". Draw a rectangle around each of these points on the original image and label them.

Solution:
The change for part (a) is straightforward … just use a Gaussian for w. The sigma for the Gaussian is up to you to pick. Our author states that sigma = 2.0 gives good results.

For part (b), we know that the eigenvalues of a 2x2 symmetric matrix are

$$v = \frac{(a+c) \pm \sqrt{(a-c)^2 + 4b^2}}{2}$$

The smaller eigenvalue will be the one with the negative sign. You can compute the smallest eigenvalue at each point in the image using the Matlab command

```
v2 = ((A11+A22)-sqrt( (A11-A22).^2 + 4*A12.^2 ))/2;
```

For part (c), the Matlab "sort" function is handy. But you not only want to return the sorted values, you want to know the indices of the sorted points. You can do this using

```
[vals, indices] = sort(vals, 'descend');
```

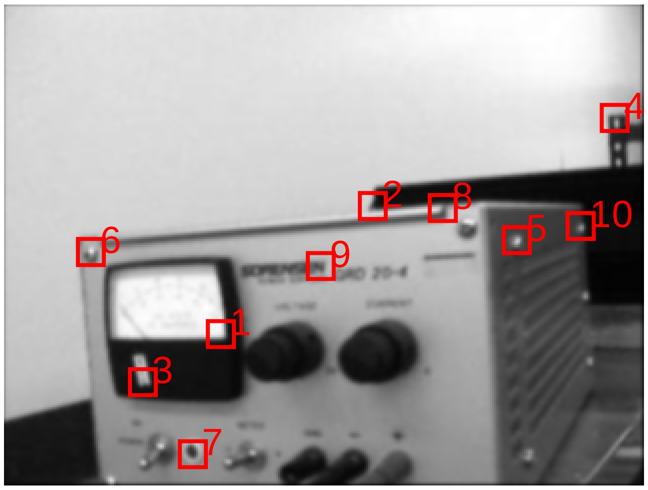
The Matlab code, and the resulting image:

```
clear all
close all

I = double(imread('test000.jpg'));

% Apply Gaussian blur
sd = 1.0;
I = imfilter(I, fspecial('gaussian', round(6*sd), sd));

imshow(I, []);

% Compute the gradient components
Gx = imfilter(I, [-1  1]);
```

[1] Do not use "for" loops to go through the image and call Matlab's "eig" function at every point. Instead, use the equation for the eigenvalue that you calculated in HW1, problem 3. You should be able to do this without any "for" loops.

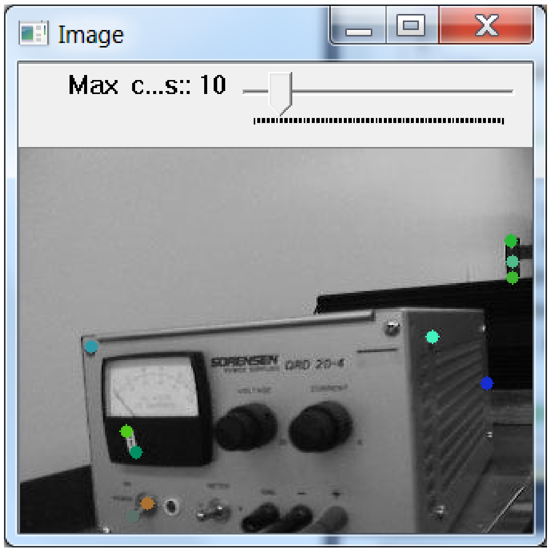
```
Gy = imfilter(I, [-1; 1]);

% Compute the products of the gradients at each pixel
Gxx = Gx .* Gx;
Gxy = Gx .* Gy;
Gyy = Gy .* Gy;

% Size of neighborhood over which to compute corner features.
N = 13;

%w = ones(N);      % The neighborhood
si = 2.0;          % Sigma for integration step
w = fspecial('gaussian', N, si);     % The neighborhood

% Sum the G's over the window size.
% Note:  these convolutions can be expensive for large window sizes.
% If time is critical, you can always do two 1D convolutions (row
% first, then column) since the mask is separable.  For really
% large windows, do the convolution in the Fourier domain.
A11 = imfilter(Gxx, w);
A12 = imfilter(Gxy, w);
A22 = imfilter(Gyy, w);

% At each pixel (x,y), we have the 2x2 matrix
%   [A11(x,y)   A12(x,y);
%    A21(x,y)   A22(x,y)]
% Of course, A21 = A12.

% Find the eigenvalues of A.  These satisfy the equation Ax = vx, where v
% is an eigenvalue and x is the corresponding 2x1 eigenvector.
% We can solve by taking (A - vI)x = 0, and so we find v such that
% det(A - vI) = 0.

% The two eigenvalues (actually all we need is the v2 value)
% v1 = ((A11+A22)+sqrt( (A11-A22).^2 + 4*A12.^2 ))/2;
v2 = ((A11+A22)-sqrt( (A11-A22).^2 + 4*A12.^2 ))/2;
s = v2;

% Choose a suppression radius, for non-maxima suppression
r = N;

% Find local maxima within each neighborhood of radius 2r
Lmax = (s==imdilate(s, strel('disk',2*r)));

% Note - we don't want to detect points too close to the border, so just
% zero out everything near the border.
Lmax(1:N,:) = false;
Lmax(:,1:N) = false;
Lmax(end-N:end,:) = false;
Lmax(:,end-N:end) = false;

% Get a list of the indices of all the potential interest points
[rows cols] = find(Lmax);
```

```
% Get the values of those interest points
vals = s(Lmax);

% Sort in descending order.  "vals" are the sorted values; "indices" are
% the corresponding indices of those values.
[vals, indices] = sort(vals, 'descend');

% Draw a box around the interest points, of size NxN
for i=1:10
    x = cols(indices(i));
    y = rows(indices(i));

    rectangle('Position', [x-N/2 y-N/2 N N], ...
        'EdgeColor', 'r', ...
        'Linewidth', 1.5);      % default is 0.5

    text(x+5,y-5, sprintf('%d', i), ...
        'Color', 'r', ...         % label with id number
        'FontSize', 14);          % default is 10
end
```



4. **(15 pts) Run the OpenCV version of the Shi-Tomasi corner detector, which is implemented in the function "`goodFeaturesToTrack`". The use of the Shi-Tomasi corner detector is illustrated in a tutorial on the http://docs.opencv.org/ website.**
   a. **Apply the program to the image "test000.jpg" and find the top 10 corners (note - you may not get exactly the same corners as your Matlab program finds in the previous problem). Give the code you used and the resulting image.**
   b. **Apply the program to the image "cube1.jpg". See if you can find all the corners on the checkerboard pattern (you will have change the program to increase the maximum allowable number corners to find).**

Solution:
(a) I just ran the tutorial from the website, except that I used the image "test000.jpg". Here is the result:

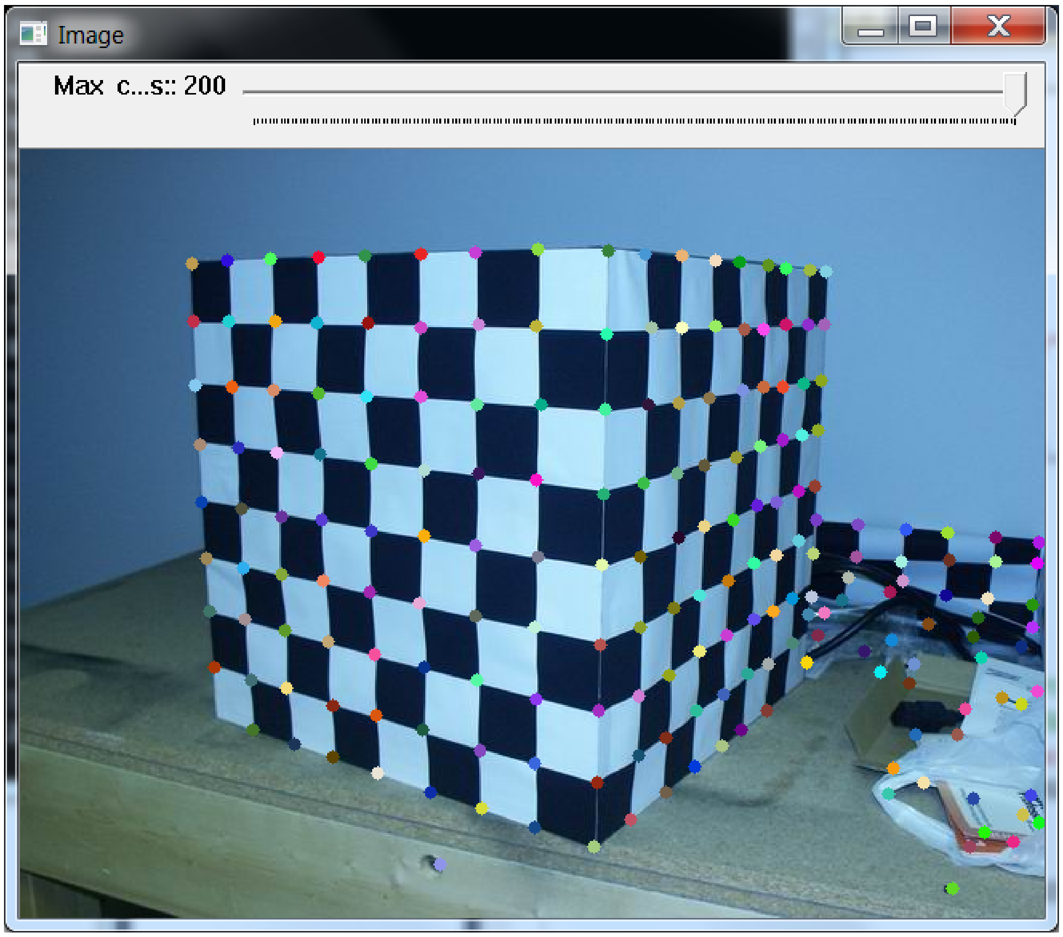


And the code:

```
/**
 * @function goodFeaturesToTrack_Demo.cpp
 * @brief Demo code for detecting corners using Shi-Tomasi method
 * @author OpenCV team
 */

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

using namespace cv;
using namespace std;

/// Global variables
Mat src, src_gray;

int maxCorners = 23;
int maxTrackbar = 100;

RNG rng(12345);
const char* source_window = "Image";

/// Function header
void goodFeaturesToTrack_Demo( int, void* );

/**
 * @function main
 */
int main( int, char** argv )
{
  /// Load source image and convert it to gray
  src = imread( "C:/Users/whoff/Documents/Teaching/Eggn512/hw/hw3/test000.jpg", 1 );
  cvtColor( src, src_gray, CV_BGR2GRAY );

  /// Create Window
  namedWindow( source_window, CV_WINDOW_AUTOSIZE );

  /// Create Trackbar to set the number of corners
  createTrackbar( "Max  corners:", source_window, &maxCorners, maxTrackbar,
goodFeaturesToTrack_Demo );

  imshow( source_window, src );

  goodFeaturesToTrack_Demo( 0, 0 );

  waitKey(0);
  return(0);
}

/**
 * @function goodFeaturesToTrack_Demo.cpp
 * @brief Apply Shi-Tomasi corner detector
 */
void goodFeaturesToTrack_Demo( int, void* )
{
  if( maxCorners < 1 ) { maxCorners = 1; }

  /// Parameters for Shi-Tomasi algorithm
  vector<Point2f> corners;
  double qualityLevel = 0.01;
  double minDistance = 10;
  int blockSize = 3;
  bool useHarrisDetector = false;
  double k = 0.04;

  /// Copy the source image
  Mat copy;
  copy = src.clone();

  /// Apply corner detection
  goodFeaturesToTrack( src_gray,
               corners,
               maxCorners,
               qualityLevel,
               minDistance,
               Mat(),
               blockSize,
               useHarrisDetector,
               k );


  /// Draw corners detected
  cout<<"** Number of corners detected: "<<corners.size()<<endl;
  int r = 4;
  for( size_t i = 0; i < corners.size(); i++ )
     { circle( copy, corners[i], r, Scalar(rng.uniform(0,255), rng.uniform(0,255),
rng.uniform(0,255)), -1, 8, 0 ); }
```

```
  /// Show what you got
  namedWindow( source_window, CV_WINDOW_AUTOSIZE );
  imshow( source_window, copy );
}
```

(b) For the cube1.jpg image, I changed the "maxTrackbar" parameter to allow up to 200 corners. Here is the result:



5. (15 pts) Consider the 3x3 template *w* as shown.

w

-1	0	-1
0	4	0
-1	0	-1

a. Compute (by hand) the normalized cross correlation score of template *w* with image *f*, at the center position of *f*. You might want to check your answer using Matlab's normxcorr2 function.

f

1	0	1	0	1
1	2	1	1	1
0	2	6	0	0
1	1	3	2	1
1	0	1	0	1

b. Give a different image *f* such that the normalized cross correlation score of template *w* with image *f*, at the center position of *f*, yields a score of -1. Assume that the image is the type unsigned 8-bit integer (ie, its values lie between 0 and 255).

Solution:

(a) The normalized cross correlation score of a template *w* with image *f* is

$$c(x,y) = \frac{\sum_{s,t}\left[w(s,t)-\bar{w}\right]\left[f(x+s,y+t)-\bar{f}\right]}{\left\{\sum_{s,t}\left[w(s,t)-\bar{w}\right]^2 \sum_{s,t}\left[f(x+s,y+t)-\bar{f}\right]^2\right\}}$$

The mean of *w* is zero. The sum of the squared values of *w* is $\sum_{s,t}\left[w(s,t)-\bar{w}\right]^2 = 20$.

We only need to look at the center 3x3 portion of *f*. The mean of the 3x3 region of *f* at the center location is 2.0. Subtracting off the mean from that 3x3 region of *f* results in

	0	-1	-1	
	0	4	-2	
	-1	1	0	

The sum of the squared values $\sum_{s,t}\left[f(x+s,y+t)-\bar{f}\right]^2 = 24$.

The numerator is $\sum_{s,t}\left[w(s,t)-\bar{w}\right]\left[f(x+s,y+t)-\bar{f}\right] = 1+16+1=18$.

So c at the center point is (18)/sqrt(20x24)=0.8216. This matches what normxcorr2 produces.

(b) An image that is identical to the template, but has opposite signs for each value, will yield a cross correlation score of -1.0. However, since we are restricted to the values that can be represented in unsigned 8-bit integers, we can't use negative numbers. But the cross-correlation operator subtracts off the mean anyway, so we can add a constant to the image to make sure the values are non-negative. So, if I add 4 to the negative of w, I get

f

	5	4	5	
	4	0	4	
	5	4	5	

Doing a normalized cross correlation of w with this image results in a score of -1.0 at the center. The other values of *f* don't matter since they are outside the boundaries of the template.

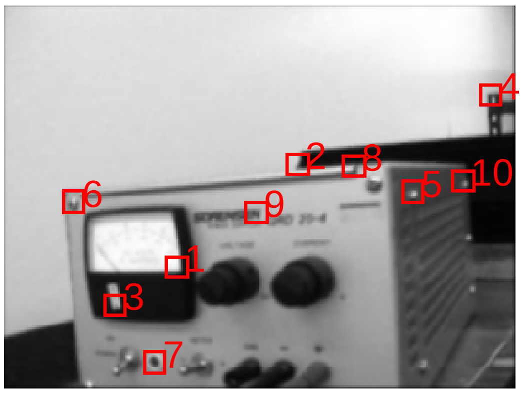
6. (20 pts) Using normalized cross correlation, match the top 10 points from the corner detector program of problem 3, from image "test000.jpg" to their best matches in image "test012.jpg". You can use Matlab's "normxcorr2" function. Mark the best matches in the second image, and label them with their identifying index from the first image (i.e., "1", "2", etc).

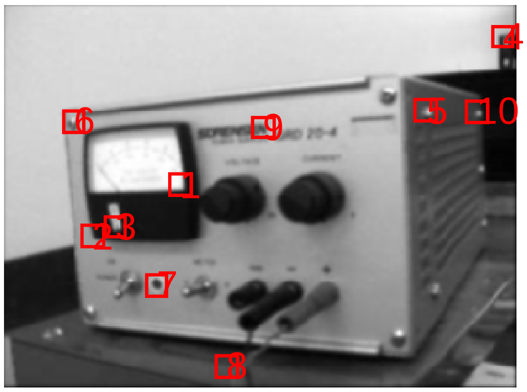
Solution:
We append the following code to the program from problem #3.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Now match these points to another image
I2 = double(imread('test012.jpg'));

% Apply Gaussian blur
I2 = imfilter(I2, fspecial('gaussian', round(6*sd), sd));

figure, imshow(I2, []);

% For each corner point found above, we will extract a template subimage of
% size NxN centered on that point, and try to match it to the second image.
for i=1:10
    x = cols(indices(i));       % Location of corner point in image 1
    y = rows(indices(i));

    % Get the template from the first image, surrounding this point
    M = floor(N/2);
    T = I(y-M:y+M, x-M:x+M);

    C = normxcorr2(T,I2);   % Do normalized cross correlation

    % The scores image C is bigger than I, by M rows and M columns along
    % the sides and the top and bottom.  So when we find the location of
    % the peak score, we should subtract M from the indices.
    cmax = max(C(:));

    [y2 x2] = find(C==cmax);
    y2 = y2-M;
    x2 = x2-M;

    fprintf('Point %d matches with score=%f\n', i, cmax);

    rectangle('Position', [x2-N/2 y2-N/2 N N], ...
        'EdgeColor', 'r', ...
        'Linewidth', 1.5);      % default is 0.5
    text(x2,y2, sprintf('%d', i), ...
        'Color', 'r', ...       % label with id number
        'FontSize', 14);        % default is 10
end
```

The output is:

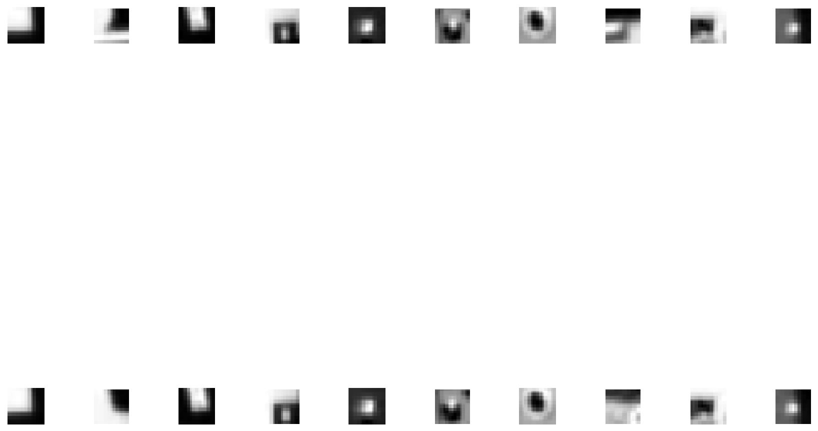
```
Point 1 matches with score=0.998280
Point 2 matches with score=0.935472
Point 3 matches with score=0.980418
Point 4 matches with score=0.964987
Point 5 matches with score=0.966965
Point 6 matches with score=0.913057
Point 7 matches with score=0.983590
Point 8 matches with score=0.906289
Point 9 matches with score=0.995221
Point 10 matches with score=0.982813
```





Looking at the results, these points matched correctly: 1,3,4,5,6,7,9,10. These were wrong: 2,8.

It is interesting to display the corresponding patches next to each other. In the figure below, the top row are the interest point patches from image one, and the bottom row are the corresponding patches extracted from image two.



```
%%%%%%%%%%%%%%%%%%%
% Just out of curiosity, display the corresponding interest point patches.
%  Top row will be from image one, bottom row from image two.
figure;
for i=1:10
    x = cols(indices(i));       % Location of corner point in image 1
    y = rows(indices(i));

    % Get the template from the first image, surrounding this point
    M = floor(N/2);
    T = I(y-M:y+M, x-M:x+M);
    subplot(2,10,i), imshow(T,[]);

    C = normxcorr2(T,I2);   % Do normalized cross correlation

    % The scores image C is bigger than I, by M rows and M columns along
    % the sides and the top and bottom.  So when we find the location of
    % the peak score, we should subtract M from the indices.
    cmax = max(C(:));

    [y2 x2] = find(C==cmax);
    y2 = y2-M;
    x2 = x2-M;

    subplot(2,10,i+10), imshow(I2(y2-M:y2+M, x2-M:x2+M), []);
end
```