CSE252A Computer Vision Homework 4

Yingyan Hua & Yanli Wang

Question 1. Dense Optical Flow

We implemented the single-scale Lucas-Kanade optical flow algorithm as explained in Lecture 18. In this method, we assume that the image brightness constancy equation yields a good approximation of the normal component of the motion field and that the latter is well approximated by a constant vector field within any small patch of the image plane. Namely, the local image flow (velocity) (u, v) must satisfy the following equations at each pixel within the patch: $I_x u + I_y v = -I_t$. Built upon this, we solve for (u,v) at each pixel in a window by minimizing the sum-squared error of the brightness constancy equations: E(u,v) = $\sum_{x,y\in\Omega}(I_x(x,y)u + I_y(x,y)v + I_t)^2$. This equation can be transformed into the following:

$$\begin{pmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{pmatrix} = \begin{pmatrix} -\sum I_x I_t \\ -\sum I_y I_t \end{pmatrix}$$

.

Solving this equation for the three sets of images, we got the optical flow results shown in Figs. 1-3. It's worth mentioning that optical flow is only valid in regions where the matrix on the left has rank 2. This is because when the 2×2 matrix is singular, the above equation cannot yield accurate velocity results. The matrix is singular when all gradient vectors point in the same direction, that is, when they are along an edge. In addition, the gradients have very small magnitude in a low texture region, so the optical flow in these regions is negligible. Therefore, when the smallest eigenvalues of the left matrix of the equation is smaller than a threshold value τ we do not compute the optical flow. We tried three windowsizes at 15, 30, and 100 pixels and different τ values. We found that when the windowsize increases, the optical flow gets more prominent on each of the three image sets. Without normalizing the elements in the matrices, we used τ values 13,000, 34,000, and 100,000 for the corridor images, 1,000, 4,000, and 10,000 for the sphere images, and 4,000, 31,000, and 100,000 for the synthetic images at windowsizes 15, 30, and 100 pixels respectively.

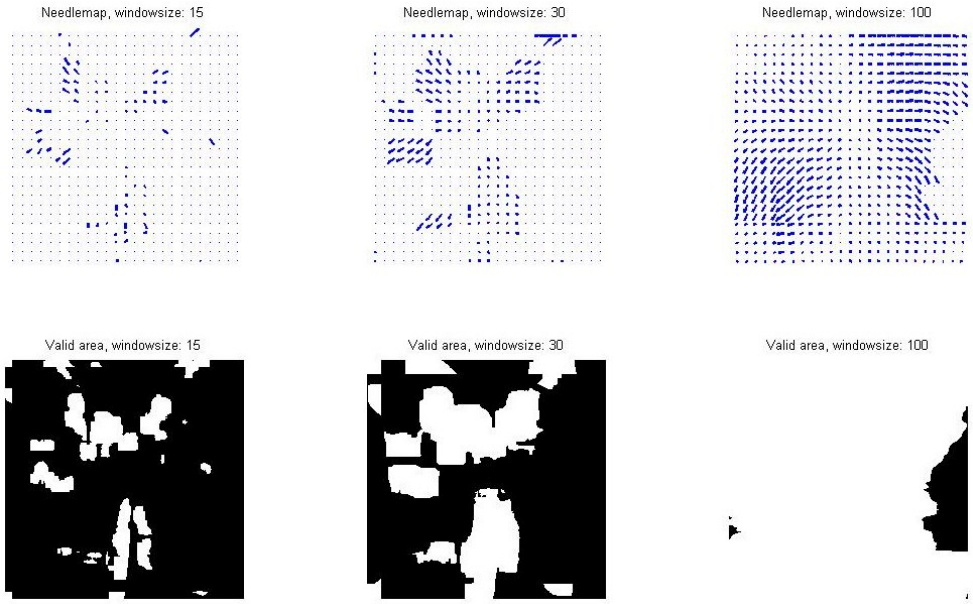

Figure 1: Dense optical flow on the corridor image when the window size is 15, 30, and 100 pixels, respectively.

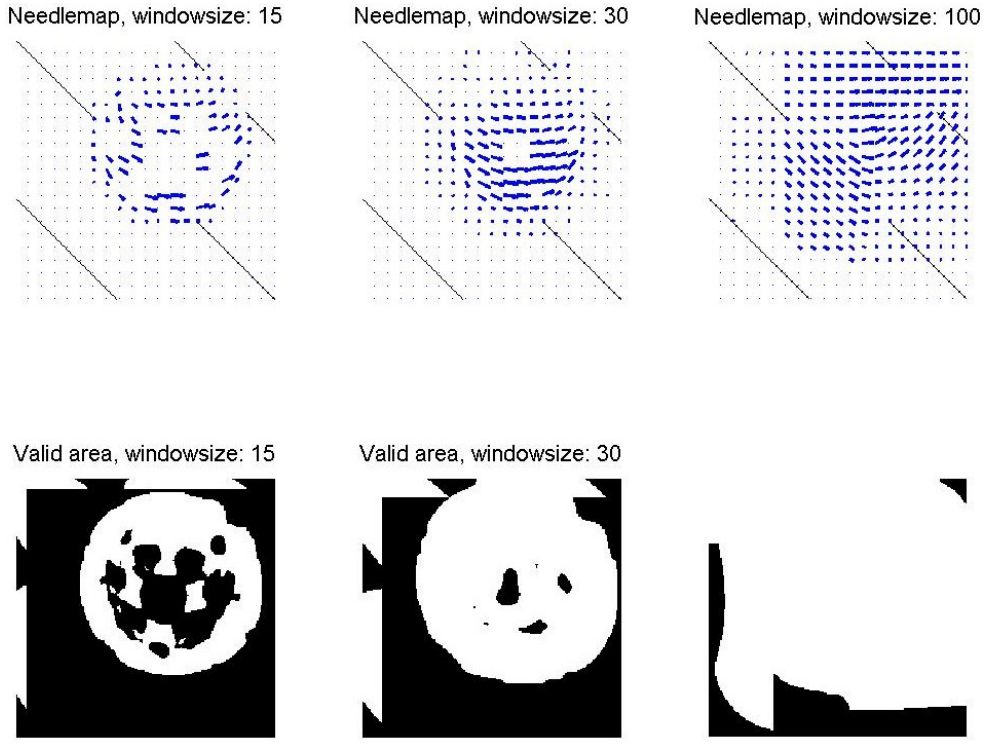

Figure 2: Dense optical flow on the corridor image when the window size is 15, 30, and 100 pixels, respectively.

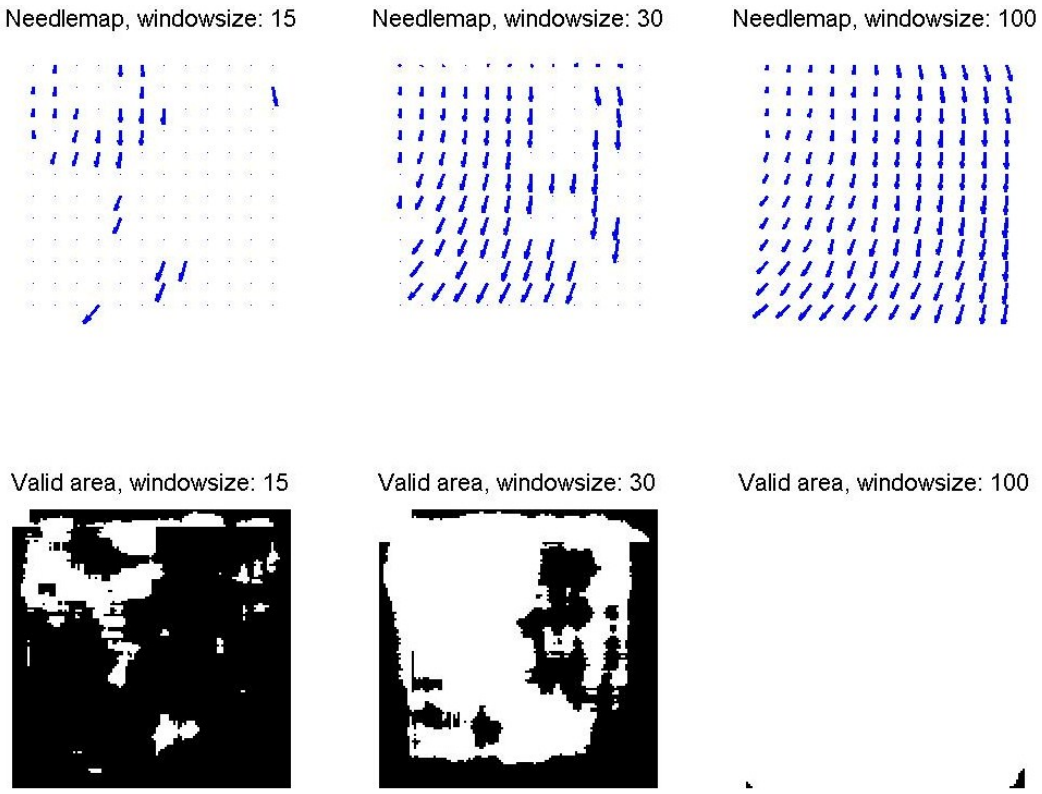

Figure 3: Dense optical flow on the corridor image when the window size is 15, 30, and 100 pixels, respectively.

Question 2. Corner Detection

We performed corner detection as instructed on the first image from each of the three image sets. The results are shown on the left of Figs. 4-6, respectively. A Gaussian smoothing kernel of standard deviation 1, and window size 7 by 7 pixels was used to smooth the images before computing the corners.

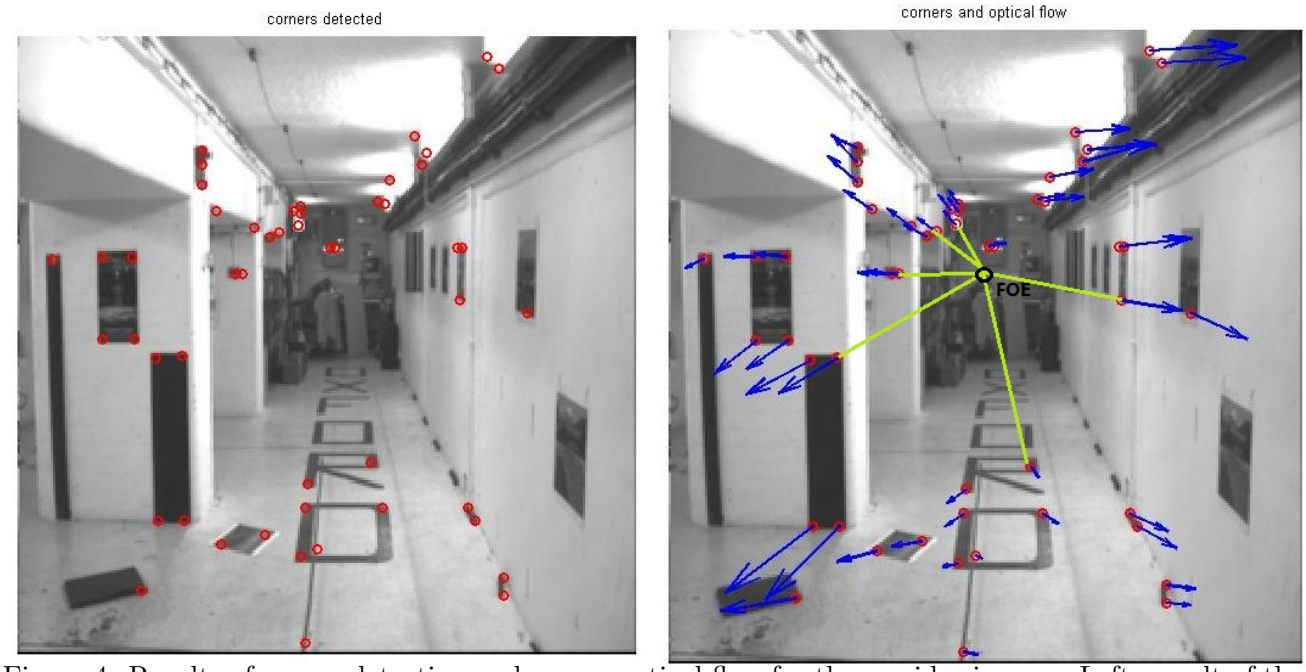

Figure 4: Results of corner detection and sparse optical flow for the corridor images. Left: result of the corner detect problem on the 1st image. Right: Result of sparse optical flow algorithm on the first two images.

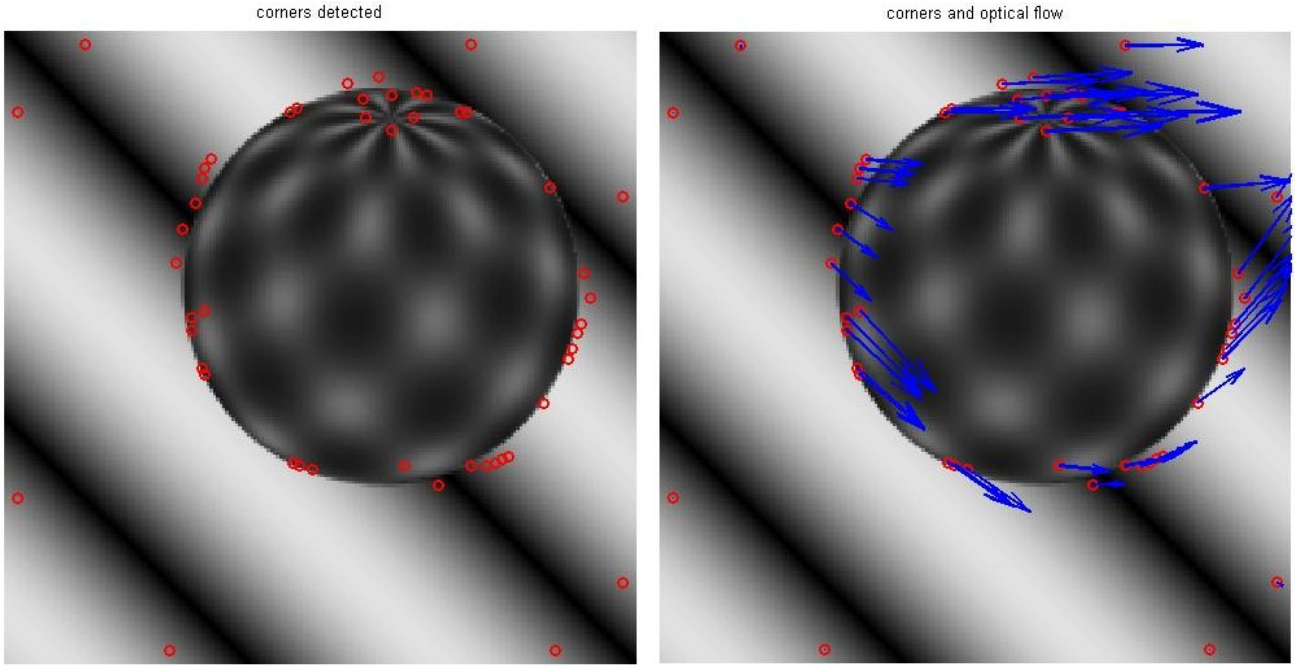

Figure 5: Results of corner detection and sparse optical flow for the sphere images. Left: result of the corner detect problem on the 1st image. Right: Result of sparse optical flow algorithm on the 1st two images.

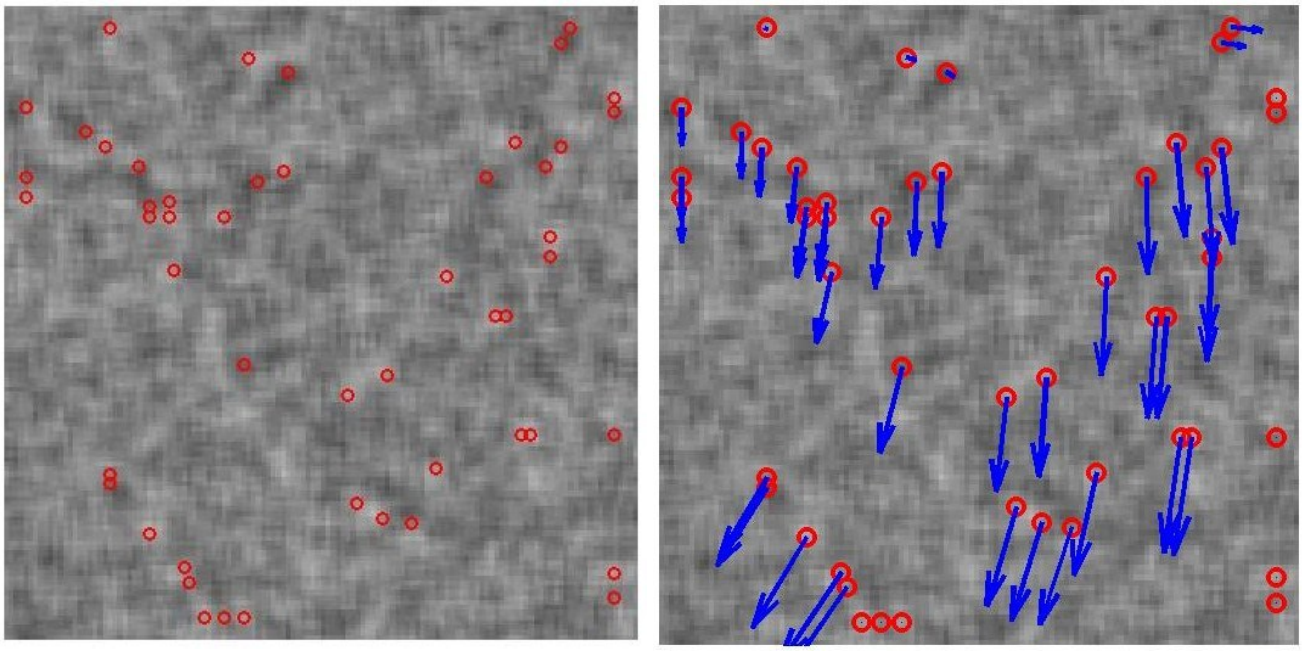

Figure 6: Results of corner detection and sparse optical flow for the synthetic images. Left: result of the corner detect problem on the first image. Right: Result of sparse optical flow algorithm on the 1st two images.

Question 3. Sparse Optical Flow

Based on results from Parts A and B, we computed the optical flow at the 50 detected corner points. Results are shown on the right of Figs. 4-6, respectively. We tried different windowsizes and threshold τ values and found that when windowsize = 100 pixels, and τ = 100,000 the sparse optical flow results are the best for the corridor and sphere images, and that when windowsize = 30 pixels, and τ = 24,000 the results are optimal for the synthetic images.

The focus of expansion (FOE) is a point in the optic flow from which all visual motion seems to emanate and which lies in the direction of forward motion. The FOE can be located from optical flow vectors alone when the motion is pure translation: it is at the intersection of the optical flow vectors. We can mark the location of the FOE in the corridor images (see Fig. 4b) because the flow vectors intersect at a particular point. If the optical flow vectors are parallel to each other, however, we assume the vectors intersect at infinity, so the FOE is at infinity. This is the case with the synthetic images. Since the flow arrows are mostly parallel, we cannot mark the exaction location of FOE on these images. In addition, when the rotational component is nonzero, the optical flow vectors do not intersect at the FOE. Therefore, we cannot locate the FOE in the sphere images because the sphere rotates in the images.

We noticed that the length of arrows is not always proportional to its distance from the FOE as we expected it to be in the sparse optical flow result for the corridor images. We observe a few arrows at the bottom right of Fig. 4b that are shorter than those above them at about the same depth. This phenomena may be accounted for by the presence of certain kind of noise in the images.

Question 4. Your Own Images

The same work as above was done on two images taken on campus. The results for corner detection and sparse optical flow are shown in Fig. 7. We find that the method works pretty well after reducing the resolution of the images. The optical flow vectors on Fig. 7b seem to cluster together at the left middle position of the image. This might be caused by the motion of the camera when taking the second image.

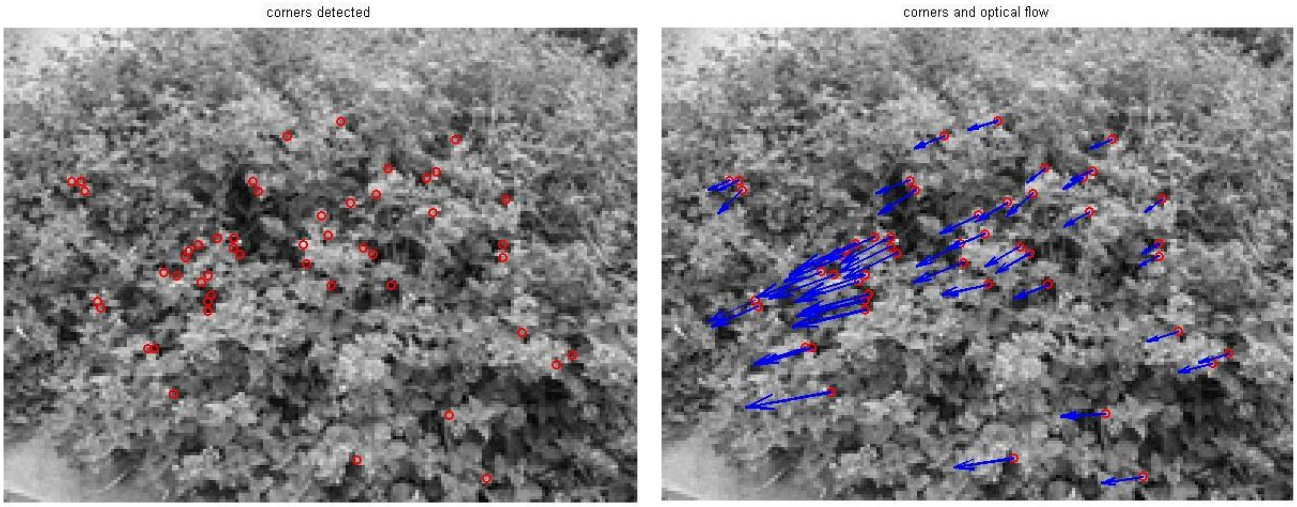

Figure 7: Results of corner detection and sparse optical flow for the synthetic images. Left: result of the corner detect problem on the first image. Right: Result of sparse optical flow algorithm on the 1st two images.

Appendix: MATLAB codes
Question 1

```
function [u, v, hitMap] = opticalFlow(I1, I2, windowSize, tau)
im1 = double(I1);
im2 = double(I2);
Ix = conv2(im1, 0.25*[-1 1; -1 1]) + conv2(im2, 0.25*[-1 1; -1 1]);
Iy = conv2(im1, 0.25*[-1 -1; 1 1]) + conv2(im2, 0.25*[-1 -1; 1 1]);
It = im2 - im1;
Ix = Ix(1:size(Ix, 1)-1, 1:size(Ix, 2)-1);
Iy = Iy(1:size(Iy, 1)-1, 1:size(Iy, 2)-1);
u = zeros(size(im1));
v = zeros(size(im2));
sz = size(im1);
hitMap = zeros(sz(1,1), sz(1,2));
sumFilter = ones(windowSize);
sumIx2 = conv2((Ix.^2), sumFilter, 'same');
sumIy2 = conv2((Iy.^2), sumFilter, 'same');
sumIxy = conv2((Ix.*Iy), sumFilter, 'same');
sumIxt = conv2((Ix.*It), sumFilter, 'same');
sumIyt = conv2((Iy.*It), sumFilter, 'same');
q1 = sumIx2;
q2 = sumIy2;
q3 = sumIxy;
S = sqrt(-4.*(q1.*q2-q3.^2) + (q1 + q2).^2);
l1 = (q1+q2+S)/2;
l2 = (q1+q2-S)/2;
for i = 1:sz(1,1)
for j = 1:sz(1,2)
if (l1(i,j) > tau && l2(i,j) > tau)
hitMap(i,j) = 1;
A = [sumIx2(i,j) sumIxy(i,j);
sumIxy(i,j) sumIy2(i,j)];
B = [sumIxt(i,j); sumIyt(i,j)];
U = inv(A)*B;
```

```
u(i,j) = (-1)*U(1);
v(i,j) = (-1)*U(2); end
end
end
[x y] = meshgrid(1:size(im1,1), 1:size(im1,2));
for i = 1:floor(size(im1,1)/10)
for j = 1:floor(size(im1,1)/10)
xs(i,j) = x(10*i, 10*j);
ys(i,j) = y(10*i, 10*j);
us(i,j) = u(10*i, 10*j);
vs(i,j) = v(10*i, 10*j);
end
end
tmp = imread('bt.000.png');
subplot(2,3,1), imshow(double(tmp)), hold on;
quiver(xs, ys, us, vs, 0.8, 'linewidth', 1.5);
```

Question 2

```
function [corners Ix Iy] = CornerDetect(img, numCorners, sigma, windowSize)
if nargin < 3, sigma = 2; end
if nargin < 4, windowSize = sigma*6+1; end
gaussianWidth = 3*sigma;
x = -gaussianWidth:1:gaussianWidth;
gaussianKernel = 1/(sqrt(2*pi)*sigma)*exp(-.5*(x.^2)/sigma^2);
smoothedXDerivativeKernel = conv2(gaussianKernel, [-1 0 1]);
smoothedYDerivativeKernel = smoothedXDerivativeKernel';
Ix = conv2(img, smoothedXDerivativeKernel, 'same');
Iy = conv2(img, smoothedYDerivativeKernel, 'same');
sumIxsqr = conv2(Ix.^2, ones(windowSize), 'same');
sumIysqr = conv2(Iy.^2, ones(windowSize), 'same');
sumIxIy = conv2(Ix.*Iy, ones(windowSize), 'same');
traceC = sumIxsqr + sumIysqr;
determinantC = sumIxsqr.*sumIysqr - sumIxIy.*sumIxIy;
lambda2 = .5*(traceC - sqrt(traceC.XOR2 - 4*determinantC));
nmsCorners = ones(size(lambda2));
neighbors = [-1 -1; -1 0; -1 1; 0 -1; 0 1; 1 -1; 1 0; 1 1]+2;
for i=1:size(neighbors,1)
k = zeros(3);
k(neighbors(i,1),neighbors(i,2)) = 1;
nmsCorners = nmsCorners & lambda2 > conv2(lambda2, k, 'same');
end
[y x] = ind2sub(size(nmsCorners);
```

```
find(nmsCorners));
corners = [x y];
hs = (windowSize-1)/2;
valid = (x > 1+hs) & (y > 1+hs) & (x < size(img,2)-hs) & (y < size(img,1)-hs);
corners = corners(valid,:);
scores = lambda2(sub2ind(size(lambda2),corners(:,2),corners(:,1)));
[v inds] = sort(-scores);
numCorners = min(length(inds),numCorners);
corners = corners(inds(1:numCorners),:);
figure(1), imshow(img), hold on
plot(corners(:,1), corners(:,2), 'ro', 'MarkerSize', 7, 'linewidth',2), title('corners detected');
end
```

Question 3

```
function c = sparseOpticalFlow(I1, corners, nCorners, u, v)
c=ones(nCorners,2);
for i=1:nCorners
c(i,1)=u(corners(i,2),corners(i,1));
c(i,2)=v(corners(i,2),corners(i,1));
end
figure;
imshow(I1);
hold on;
plot(corners(:,1), corners(:,2), 'ro', 'MarkerSize', 7, 'linewidth',2), title('corners and optical flow');
hold on;
quiver(corners(:,1),corners(:,2),c(:,1),c(:,2),1,'linewidth',2);
```