Gupta, Ajitesh
Choi, Jean
Mavalankar, Aditi Ashutosh
Department of Electrical Engineering and Computer Science
University of California, San Diego

November 23, 2016

HOMEWORK 3

Problem 1. **Image warping and merging** : Write files computeH.m and warp.m that can be used in the following skeleton code. warp takes as inputs the original image, corners of an ad in the image, and the homography H. Note that the homography should map points from the destination image to the original image, that way you will avoid problems with aliasing and sub-sampling effects when you warp. You may find the following MATLAB files useful: meshgrid, inpolygon, fix, interp2.

Report For three of the ads in stadium.jpg, run the skeleton code and include the output images in your report.

Solution

First, we have to find the homography, H, such that $x_i' = Hx_i$.

Since x_i' and Hx_i are the vectors with the same direction, the cross product would be zero.

Thus $x_i' \times Hx_i = 0$.

Denote jth row of H as h^{j^T}, then:

$$Hx_i = \begin{bmatrix} h^{1^T} x_i \\ h^{2^T} x_i \\ h^{3^T} x_i \end{bmatrix} \tag{0-1}$$

Denote the transpose of x_i' as $x_i'^T = (x_i', y_i', w_i')^T$, then $x_i' \times Hx_i = 0$ is:

$$x_i' \times Hx_i = \begin{bmatrix} y_i' h^{3^T} x_i - w_i' h^{2^T} x_i \\ w_i' h^{1^T} x_i - x_i' h^{3^T} x_i \\ x_i' h^{2^T} x_i - y_i' h^{1^T} x_i \end{bmatrix} \tag{0-2}$$

Since $h^{j^T} x_i = x_i^T h^i$, above can be rewritten as follows:

$$\begin{bmatrix} 0^T & -w_i' x_i^T & y_i' x_i^T \\ w_i' x_i^T & 0^T & -x_i' x^T \\ -y_i' x_i^T & x_i' x^T & 0^T \end{bmatrix} \begin{bmatrix} h^1 \\ h^2 \\ h^3 \end{bmatrix} = 0 \tag{0-3}$$

(0-3) has the form of $A_i h = 0$ whereas A_i is a 3×9 matrix and h is a 9×1 matrix.

A_i has the rank of 2.

Let A be the collective matrix of A_1, A_2, A_3, and A_4, i.e. four points.

Then $Ah = 0$ whereas A is a 12×9 matrix.

Since A_i has the rank of 2, A has the rank of 8 and thus has 1D nullspace which gives solution for h.

Assume the scale factor is 1 such that $||h|| = 1$.

Then h can be given by V whereas $A = U \sum V^T$, i.e. SVD of A.

Figure 1: First output image

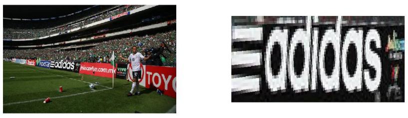

Figure 2: Second output image

Figure 3: Third output image

Problem 2. **Optical Flow** In this problem you will implement the Lucas-Kanade algorithm for computing a dense optical flow field at every pixel. You will then implement a corner detector and combine the two algorithms to compute a flow field only at reliable corner points. Your input will be pairs or sequences of images and your algorithm will output an optical flow field (u,v). Three sets of test images are available from the course website. The first contains a synthetic (random) texture, the second a rotating sphere, and the third a corridor at Oxford university. Before running your code on the images, you should first convert your images to grayscale and map intensity values to the range [0,1]. I use the synthetic dataset in the instructions below. Please include results on all three datasets in your report. For reference, your optical flow algorithm should run in seconds if you vectorize properly (for example, the eigenvalues of a 2x2 matrix can be computed directly). Again, no points will be taken off for slow code, but it will make the experiments more pleasant to run.

2.1. **Dense Optical Flow** Implement the single-scale Lucas-Kanade optical flow algorithm. This involves finding the motion (u,v) that minimizes the sum-squared error of the brightness constancy equations for each pixel in a window. As a reference, read pages 191-198 in Introductory Techniques for 3-D Computer Vision by Trucco and Verri4. Your algorithm will be implemented as a function with the following inputs,

```
function [u, v, hitMap] = opticalFlow(I1, I2, windowSize, tau)
```

Here, u and v are the x and y components of the optical flow, hitMap a binary image indicating where the corners are valid (see below), I1 and I2 are two images taken at times $t = 1$ and $t = 2$ respectively, windowSize is the width of the window used during flow computation, and τ is the threshold such that if the smallest eigenvalue of $A^T A$ is smaller than τ, then the optical flow at that position should not be computed. Recall that the optical flow is only valid in regions where

$$A^T A = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_y I_x & \sum I_y^2 \end{bmatrix} \tag{0-4}$$

A typical value for τ is 0.01. Using this value of τ, run your algorithm on all three image sets (the first two images of each set), for three different windowsizes of your choice. Also provide some comments on performance, impact of windowsize etc.

Solution

We see that as the window size increases the estimation of optical flow gets better as it is able to incorporate more context. This is evident from the hitmap also.

Corridor:

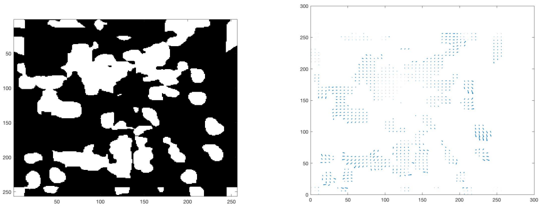

Figure 4: corridor, 15 (a) hitmap (b) flow

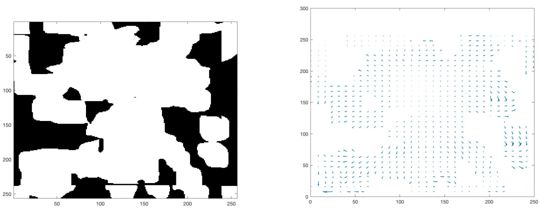

Figure 5: corridor, 25 (a) hitmap (b) flow

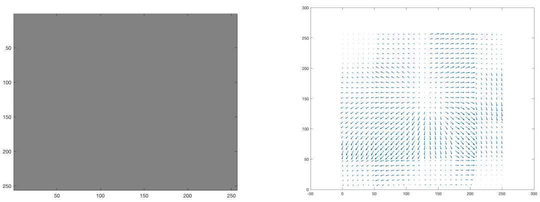

Figure 6: corridor, 100 (a) hitmap (b) flow

Sphere:

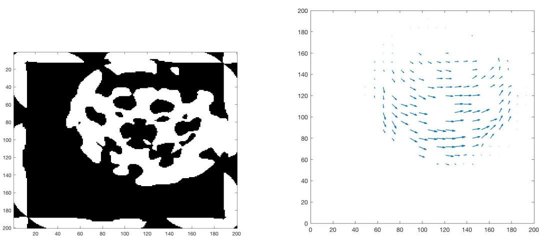

Figure 7: sphere, 15 (a) hitmap (b) flow

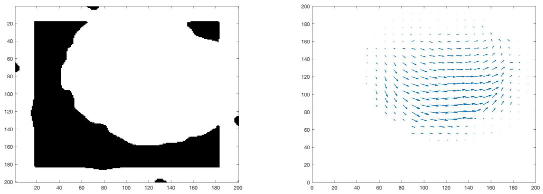

Figure 8: sphere, 25 (a) hitmap (b) flow

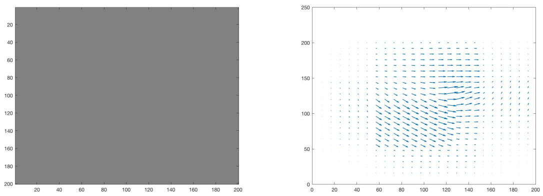

Figure 9: sphere, 100 (a) hitmap (b) flow

Synthesized:

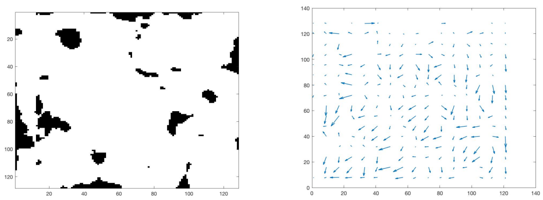

Figure 10: synth, 15 (a) hitmap (b) flow

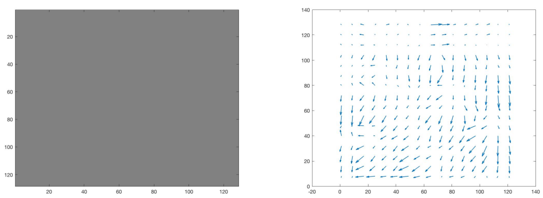

Figure 11: synth, 25 (a) hitmap (b) flow

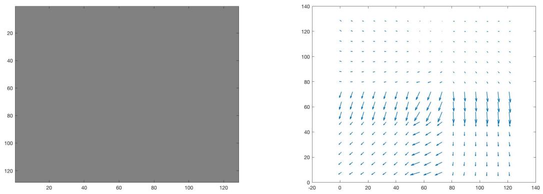

Figure 12: synth, 100 (a) hitmap (b) flow

2.2. **Corner Detection** Use your corner detector from Assignment 2 to detect 50 corners in the provided images. Use a smoothing kernel with standard deviation 1, and windowsize of 7 by 7 pixels for your corner detection throughout this assignment. Include a image similar to Fig. 4a in your report. If you were unable to create a corner detection algorihtm in the previous assignment, please email the TA for code.

Solution

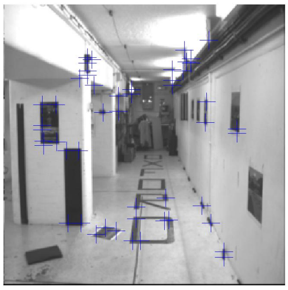

Figure 13: corridor, corners

2.3. **Sparse Optical Flow** Combine Parts A and B to output an optical flow field at the 50 detected corner points. Include result plots as in Fig. 4b. Select appropriate values for windowsize and that gives you the best results. Provide a discussion about the focus of expansion (FOE) and mark manually in your images where it is located. Is it possible to mark the FOE in all image pairs? Why / why not?

Solution 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 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.

Figure 14: corridor, flow

Problem 3. **Iterative Coarse to Fine Optical Flow** Implement the iterative coarse to fine optical flow algorithm described in the class lecture notes (pages 8 and 9 in lecture 13). Show how the coarse to fine algorithm algorithm works better on the first two frames inside of flower.zip than dense optical flow. You can do this by creating a quiver plot using your code from problem 2 and a quiver plot for the coarse to fine algorithm. Try 3 different window sizes: one of your choice, 5, and 15 pixels. Where does the dense optical flow algorithm struggle that this algorithm does better with? Can you explain this in terms of depth or movement distance of pixels? Comment on how window size affects the coarse to fine algorithm? Do you think that the coarse to fine algorithm is strictly better than the standard optical flow algorithm? Example output shown in Fig. 5a. Note: Like in problem 2, convert the image to intensity grayscale images.

Solution We see a marked difference between the output of the iterative optical flow and the dense one. The dense one is the last image in figure 15 while the others are coarse to fine output.

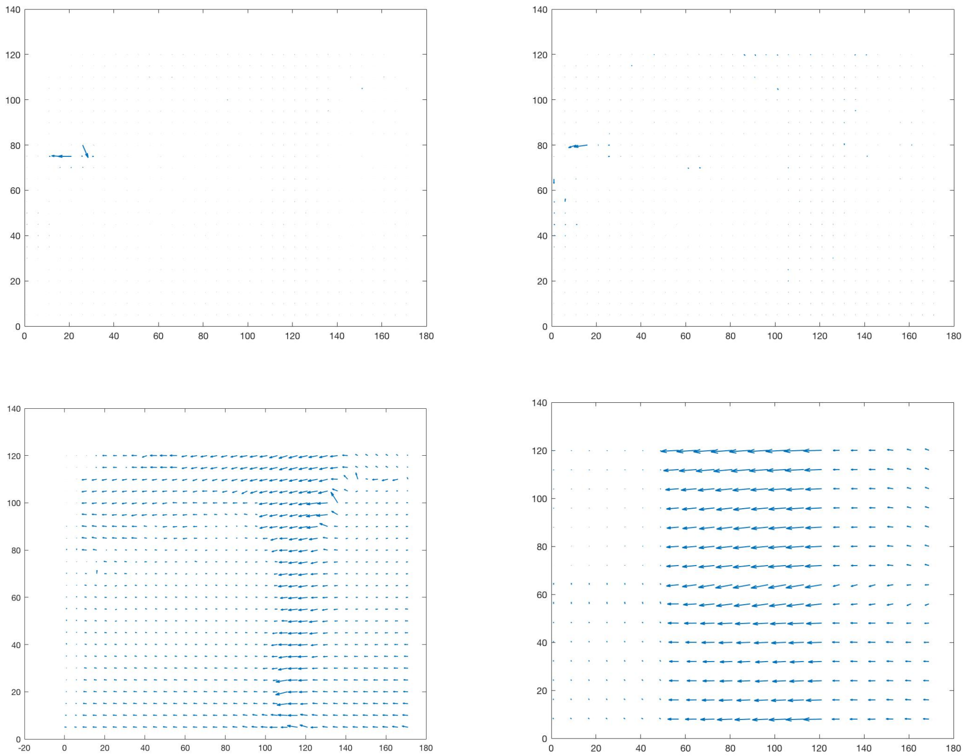

Figure 15: a) window size = 10 b) window size = 5 c) window size = 15 d) Dense optical flow

Problem 4. **Background Subtraction and Motion Segmentation**

4.1. **Background Subtraction**In this problem you will remove the dynamic portions of an image from a static background. In this case, the camera is not moving and objects within the scene or moving. For each consecutive pair of frames, background subtraction will calculate. $|I(x, y, t)I(x, y, t1)| > \tau$, where $I(x, y, t)$ is the pixel intensity of the tth frame of the image I, at position (x, y). Also τ is the threshold parameter. $|I(x, y, t)I(x, y, t1)| > \tau$, is the foreground mask for the frame at time t. By masking out the pixels that are greater than τ, you can create an estimate background for the frame. By calculating the mean of the estimated backgrounds you can create a global background for the sequence, as shown in the figure. Note: Convert the image to a gray scale image.

```
function [background] = backgrounSubtract(framesequence, tau)
```

The variable framesequence can be a string representing a directory of frame images or cell array of frames or any other reasonable input. Run your code on the highway and truck sequence and include the backgrounds for each sequence as a figure.

Solution We used a threshold value of 10 for the assignment.

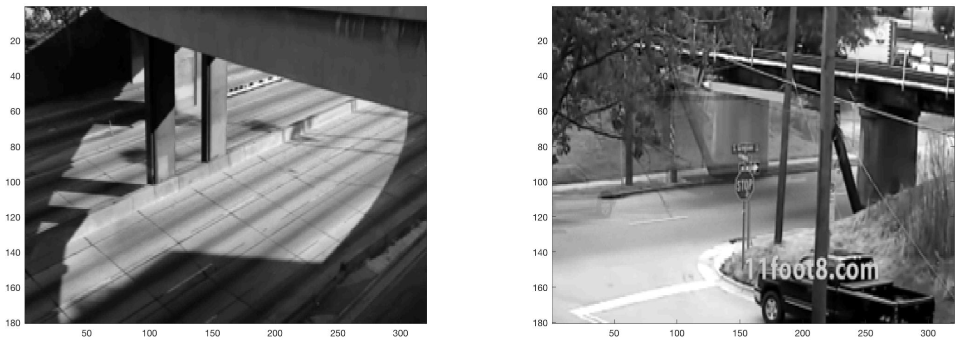

Figure 16: (a) Highway (b) Truck

4.2. **Motion Seqmentation** The previous algorithm only works on static cameras and a stable background. However, it is also possible to do a similar segmentation using motion cues, however it is much harder. Using the outputs of your iterative coarse to fine optical flow algorithm, can you segment out the tree from the rest of the first frame of the flower sequence? Hint: the magnitudes of the motion vectors at different depths can be quite different. Provide a figure of the segmented tree (an image with just the tree in it) and explain how you were able to do this. Note: Like in each problem convert the image to gray scale image.

Solution

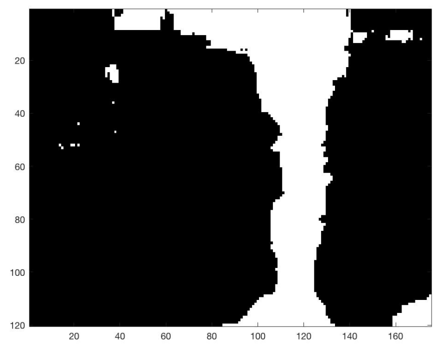

Figure 17: (a) Tree

We calculate the total motion at each point by calculating the norm of the individual motion vectors from iterative coarse to fine refinement and then we threshold on those values. We found 1.8 to be a good threshold.

Problem 5. **Hough Lines** In this problem, you will implement a Hough Transform method for finding lines. For the algorithm, refer lecture 12. You may use the inbuilt matlab functions for detecting the edges. However, keep in mind that you may need to experiment with other parameters till you get the expected edges from the given image (Refer matlab documentation of edge function for more details). You should not use any of the inbuilt Hough methods.//

- Produce a simple 11×11 test image made up of zeros with 5 ones in it, arranged like the 5 points. Compute and display its Hough Transform. Threshold the HT by looking for any (ρ, θ) cells that contains more than 2 votes then plot the corresponding lines in (x, y)-space on top of the original image.

  Solution

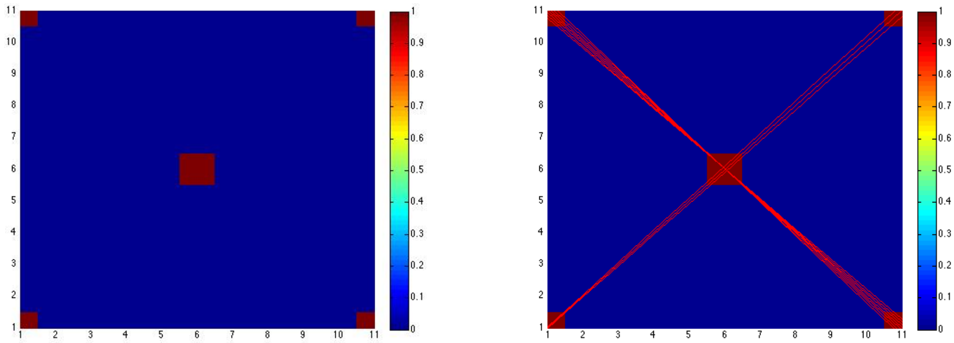

Figure 18: (a) Original 11×11 image (b) Image after applying Hough Transform

- Load in the image lane.png. Compute and display its edges using the Sobel operator with your threshold settings similar to that in HW2. Now compute and display the HT of the binary edge image E. As before, threshold the Hough Transform and plot the corresponding lines atop the original image; this time, use a threshold of 75% over the entire HT, i.e. $0.75 \times max(HT(:))$.

  Solution

Figure 19: (a) After applying sobel operator (b) After applying Hough transform

- Now that you have mastered Hough transform, Repeat the procedure for MS Commons room images (common1.jpg and common2.jpg). Set the appropriate threshold parameters to plot corresponding lines atop the original image. You will be graded on the accuracy of the result.

  Solution

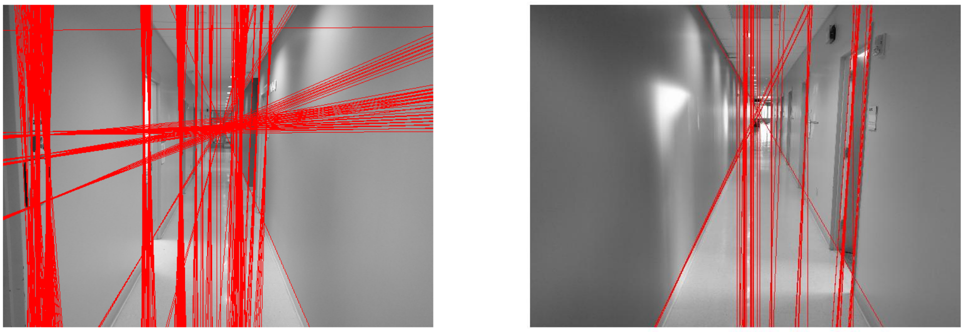

Figure 20: (a) Master's Commons I (b) Master's Commons II

Explanation

The thresholding parameters for the Hough transform need to be decided very carefully, as they can lead to either non-detection of important edges of detection of spurious edges. This is why tweaking of the parameters to find the important edges is important in this method.

In our implementation, we first tested the method on a sample 11×11 image as shown in the above figures. We then applied this algorithm to the 'Lanes.png' image and the results were generated as shown. Here, the threshold was 75% of the maximum value in the H matrix.

We then moved on to the Master's commons images. Here, we used a lower threshold parameter and reduced the sampling on theta, in order to reduce the spurious edges that we observed. The limits of the threshold that we tested were $15\% - 60\%$ and the best results that we obtained are in the figures shown above. For a higher threshold, the significant edges i.e. the ones converging into the corridor were missing and for a very low threshold, there were spurious edges.

Appendix

```
I1 = imread('stadium.jpg');

% get points from the image
figure(10)
imshow(I1)

% select points on the image, preferably the corners of an ad.
points = ginput(4);
figure(1)
subplot(1,2,1);
imshow(I1);

% choose your own set of points to warp your ad too
new_points = [1,1,1;...
              400,1,1;...
              400,200,1;...
              1,200,1];
H = computeH(points, new_points);

% warp will return just the ad rectified
```

```
warped_img = warp(I1, new_points, H);
subplot(1,2,2);
imshow(warped_img);
```

Listing 1: MATLAB source for Question 1

```
function [H] = computeH(p1, p2)
% http://www.cse.iitd.ernet.in/~suban/vision/geometry/node24.html
A = [];
p1(:,3) = 1;
zero = zeros([1,3]);
for i=1:4
    A = [A;
         zero, -p1(i,3).*p2(i,:), p1(i,2).*p2(i,:);...
         p1(i,3).*p2(i,:), zero, -p1(i,1).*p2(i,:);...
         -p1(i,2).*p2(i,:), p1(i,1).*p2(i,:), zero;];
end
[~,~,v] = svd(A);
H = v(:,end);
H = reshape(H,[3,3])';
end
```

Listing 2: MATLAB source for Computing H

```
close all;

%I1 = imread('corridor/bt.000.png');
%I2 = imread('corridor/bt.001.png');

%I1 = imread('sphere/sphere.0.png');
%I2 = imread('sphere/sphere.1.png');

I1 = imread('synth/synth_000.png');
I2 = imread('synth/synth_001.png');

%I1 = imread('flower/00029.png');
%I2 = imread('flower/00030.png');

figure,imshow(I1);
figure,imshow(I2);

if(size(I1,3)==3)
    I1 = rgb2gray(I1);
    I2 = rgb2gray(I2);
end

I1 = mat2gray(I1);
I2 = mat2gray(I2);

tau = 0.01;
windowSize = 15;

skip = 8;

[u, v, hitMap] = opticalFlowLK1(I1, I2, windowSize, tau);
[x, y] = meshgrid(1:skip:size(I1,2), size(I1,1):-skip:1);
qu = u(1:skip:size(I1,1), 1:skip:size(I1,2));
qv = v(1:skip:size(I1,1), 1:skip:size(I1,2));
quiver(x,y, qu, -qv,'linewidth', 1);
figure,imagesc(hitMap),colormap(gray);
```

Listing 3: MATLAB source for Question 2 part 1

```matlab
function [u, v, hitMap] = opticalFlowLK1(I1, I2, windowSize, tau, points)
% https://www.mathworks.com/help/vision/ref/opticalflowlk-class.html

[h,w] = size(I1);
hitMap = zeros([h,w]);
u = zeros([h,w]);
v = zeros([h,w]);

%% Smoothing with gaussian kernel
gaussian = fspecial('gaussian', [11,11], 4);
I1 = imfilter(I1, gaussian);
I2 = imfilter(I2, gaussian);
window = ones(windowSize);

%% Calculating gradient images
d = 1/12.*[-1,8,0,-8,1];
gx = conv2(I1, d, 'same');
gy = conv2(I1, d', 'same');
gt = I2-I1;

%% Calculating product of derivatives
Ixx = gx.*gx;
Iyy = gy.*gy;
Ixy = gx.*gy;
Ixt = gx.*gt;
Iyt = gy.*gt;

%% Calculating weighted sum of product of derivatives
Sxx = conv2(Ixx, window, 'same');
Syy = conv2(Iyy, window, 'same');
Sxy = conv2(Ixy, window, 'same');
Sxt = conv2(Ixt, window, 'same');
Syt = conv2(Iyt, window, 'same');

area = (windowSize*windowSize);

if nargin==4
    for i=1:h
        for j=1:w
            A = [Sxx(i,j) Sxy(i,j);...
                 Sxy(i,j) Syy(i,j)];
            e = eig(A);
            if(all(e>tau))
                hitMap(i,j) = 255;
                B = -1.*[Sxt(i,j);...
                         Syt(i,j)];
                motion = A\B;
                u(i,j) = motion(1);
                v(i,j) = motion(2);
            end
        end
    end
else
    for k=1:50
        i = points(k,1);
        j = points(k,2);
        A = [Sxx(i,j) Sxy(i,j);...
             Sxy(i,j) Syy(i,j)];
        e = eig(A);
        if(all(e>tau))
            hitMap(i,j) = 255;
            B = -[Sxt(i,j);...
```

```
                  Syt(i,j)];
            motion = A\B;
            u(i,j) = motion(1);
            v(i,j) = motion(2);
        end
    end
end
```

Listing 4: MATLAB source for Question 2 part 1

```
%I = imread('corridor/bt.001.png');
%I = imread('flower/00029.png');

if(length(size(I))==3)
    I = rgb2gray(I);
end

smoothSTD = 1;
windowSize = 7;
nCorners = 50;

[corners] = CornerDetect(I, nCorners, smoothSTD, windowSize);

figure, imshow(I);
hold on;
for i=1:nCorners
plot(corners(i,2),corners(i,1),'b+','MarkerSize',20);
end
```

Listing 5: MATLAB source for Question 2 part 2

```
%% Preprocessing

I1 = imread('corridor/bt.000.png');
I2 = imread('corridor/bt.002.png');

%I1 = imread('sphere/sphere.0.png');
%I2 = imread('sphere/sphere.5.png');

%I1 = imread('synth/synth_000.png');
%I2 = imread('synth/synth_001.png');

figure,imshow(I1);
figure,imshow(I2);
if(size(I1,3)==3)
    I1 = rgb2gray(I1);
    I2 = rgb2gray(I2);
end


%% Corner Detection

smoothSTD = 1;
windowSize = 7;
nCorners = 50;

[corners] = CornerDetect(I1, nCorners, smoothSTD, windowSize);

figure, imshow(I1);
hold on;
for i=1:nCorners
plot(corners(i,2),corners(i,1),'b+','MarkerSize',20);
```

```
end

%% Optical flow estimation

I1 = im2double(I1);
I2 = im2double(I2);

tau = 0.01;
windowSize = 100;

[u, v, hitMap] = opticalFlowLK1(I1, I2, windowSize, tau, corners);
x = 1:1:size(I1,2);
y = 1:1:size(I1,1);
[X,Y] = meshgrid(x,y);

figure,imshow(hitMap);
figure,imshow(I1),hold on,quiver(X,Y,u(y,x),v(y,x),10);
```

Listing 6: MATLAB source for Question 2 part 3

```
%% Preprocess

I1 = imread('flower/00029.png');
I2 = imread('flower/00030.png');

if size(I1,3)==3
    I1 = rgb2gray(I1);
end

if size(I2,3)==3
    I2 = rgb2gray(I2);
end

%%  Spatial pyramid
ia = cell(3,1);
ia{3} = I1;
ia{2} = impyramid(I1, 'reduce');
ia{1} = impyramid(ia{2}, 'reduce');

ib = cell(3,1);
ib{3} = I2;
ib{2} = impyramid(I2, 'reduce');
ib{1} = impyramid(ib{2}, 'reduce');

%% Parameters
iters = 10;
tau = 0.01;
windowSize = 15;

%% motion estimate at the lowest level of pyramid
ui=zeros(size(ia{1}));
vi=zeros(size(ia{1}));
[u, v] = iterOpticalFlowLK(ia{1}, ib{1}, ui , vi, windowSize, tau, iters);

%% correction of estimate while scaling up
for i=2:3
    upu = imresize(u, size(ia{i}));
    upv = imresize(v, size(ia{i}));
    upu = 2.*upu;
    upv = 2.*upv;
    [u, v] = iterOpticalFlowLK(ia{i}, ib{i}, upu, upv, windowSize, tau, iters) ;
end

```

```matlab
%% plotting the quiver plot
[x, y] = meshgrid(1:5:size(I1,2), size(I1,1):-5:1);
qu = u(1:5:size(I1,1), 1:5:size(I1,2));
qv = v(1:5:size(I1,1), 1:5:size(I1,2));
quiver(x,y, qu, -qv,'linewidth', 1);

%% 4.2 segmentation
% calculation total motion at each point
map = zeros(size(u));
for i=1:size(u ,1)
    for j=1:size(u ,2)
        map(i ,j) =sqrt((u(i ,j)).^2 + (v(i ,j)).^2);
    end
end
% thresholding to get the tree
map = imbinarize(map,1.8);
figure, imagesc(map),colormap(gray);
```

Listing 7: MATLAB source for Question 3

```matlab
function [uf,vf] = iterOpticalFlowLK(I1,I2,initu ,initv ,windowSize,tau,iters)

%% Preprocess the image
I1 = mat2gray(I1);
I2 = mat2gray(I2);

%% Find the derivates in x and y direction
d = 1/12.*[-1,8,0,-8,1];
sz = size(I1);
Ix = conv2(I1,d,'same');
Iy = conv2(I1,d', 'same');

%% Sum the derivates in windows
Ixx = conv2(Ix.^2, ones(windowSize),'same');
Iyy = conv2(Iy.^2, ones(windowSize),'same');
Ixy = conv2(Ix.*Iy, ones(windowSize),'same');

%% Iterative Lucas Kanade
uf = zeros(sz);
vf = zeros(sz);
half = floor(windowSize/2);
for i = 1:sz(1)
    for j =1:sz(2)
        left = j-half;
        right = j+half;
        top = i-half ;
        bottom = i+half ;
        if(left <=0)
            left = 1;
        end
        if(right>sz(2))
            right = sz(2);
        end
        if(top<=0)
            top = 1;
        end
        if(bottom>sz(1))
            bottom = sz(1);
        end
        win1 = I1(top:bottom,left:right);
        ix = Ix(top:bottom,left:right);
        iy = Iy(top:bottom,left:right);
        A = [Ixx(i,j) Ixy(i,j);...
```

```
            Ixy(i,j) Iyy(i,j)];
        r = rank(A);
        % inverse exists only if rank is 2
        if(r~=2)
            Ainv = zeros(2);
        else
            Ainv =inv(A);
        end
        u = initu(i,j);
        v = initv(i,j);
        % iterative refinement of estimate
        for iter = 1:iters
            [x,y] = meshgrid(1:size(I1,2), 1:size(I1,1));
            xp = x+u;
            yp = y+v;
            win2 = interp2(x,y,I2,xp(top:bottom,left:right),yp(top:bottom, left:right));
            it = win2-win1; ixt = it.*ix; iyt = it.*iy;
            B = -[sum(ixt(:)); sum(iyt(:))];
            U = Ainv*B;
            U(isnan(U)) = 0 ;
            u = u+U(1);
            v = v+U(2);
            % desired accuracy achieved
            if(abs(U(1))<tau && abs(U(2))<tau)
                break;
            end
        end
        uf(i,j)=u; vf(i,j)=v;
    end
end

%% Plotting the quiver plot
[x, y] = meshgrid(1:5:size(I1,2), size(I1,1):-5:1);
qu = uf(1:5:size(I1,1), 1:5:size(I1,2));
qv = vf(1:5:size(I1,1), 1:5:size(I1,2));
quiver(x,y, qu, -qv,'linewidth', 1);
end
```

Listing 8: MATLAB source for Question 3

```
foldername = 'truck';
files = dir(foldername);
files = files(3:end);
nframes = size(files,1);
framesequence = cell([1,nframes]);

for i=1:nframes
    im = imread(strcat(files(i).folder,'/',files(i).name));
    if length(size(im))==3
        im = rgb2gray(im);
    end
    framesequence{i} = im;
end

tau = 10;
background = backgroundSubtract(framesequence, tau);
figure,imagesc(background);colormap(gray);
```

Listing 9: MATLAB source for Question 4

```
img = zeros(11, 11);
```

```
img = uint8(img);
img(1,1) = 1;
img(1,11) = 1;
img(11,1) = 1;
img(11,11) = 1;
img(6,6) = 1;

hold on;

imagesc(img),colormap(jet)
colorbar

interval = 0.5;
D = [];

for i = -16:0.5:16
    D = [D i];
end


H = zeros(65, 362);
for i = 1:11
    for j = 1:11
        if img(i,j) == 1
            for theta = 1:0.5:181
                d = i * cosd(theta - 1) + j * sind(theta - 1);
                [d index] = min(abs(D - d));
                H(index, theta * 2) = H(index, theta * 2) + 1;
            end
        end
    end
end

for i = 1:65
    for j = 1:0.5:181
        if H(i, 2 * j) > 2
            if i > 32
                d = double(i - 33) / 2;
            else
                d = -1 * double(i) / 2;
            end
            th = j;
            x = 1:11;
            y = d * cscd(th - 1) - x * cotd(th - 1);
            axis([1,11,1,11]);
            plot(x, y, 'r');
        end
    end
end



hold off;
```

Listing 10: MATLAB source for Hough Transform of 11 × 11 image

```
img = imread('data/lanes.png');
img = rgb2gray(img);
imshow(img);
newimg = edge(img, 'sobel', 0.1);

newimg(1:15,:) = 0;
newimg(:, 1:15) = 0;
```

```
newimg(end-15:end, :) = 0;
newimg(:, end-15:end) = 0;
[length width] = size(newimg);
hold on;

interval = 0.5;
D = [];

maxd = ceil(sqrt(length * length + width * width));

for i = -maxd:0.5:maxd
    D = [D i];
end


H = zeros(4 * maxd + 1, 362);
for i = 1:length
    for j = 1:width
        if newimg(i,j) == 1
            for theta = 1:0.5:181
                d = i * cosd(theta - 1) + j * sind(theta - 1);
                [d index] = min(abs(D - d));
                H(index, theta * 2) = H(index, theta * 2) + 1;
            end
        end
    end
end

R = max(H);
maximum = max(R);
threshold = 3 * double(maximum) / 4;

for i = 1:4 * maxd + 1
    for j = 1:0.5:181
        if H(i, 2 * j) > threshold
            if i > 2 * maxd
                d = double(i - 2 * maxd - 1) / 2;
            else
                d = -1 * double(i) / 2;
            end
            th = j;
            x = 1:length;
            y = d * cscd(th - 1) - x * cotd(th - 1);
            plot(y, x, 'r');
        end
    end
end



hold off;
```

Listing 11: MATLAB source for Hough Transform of Lanes

```
img = imread('data/mscommons1.jpeg');
img = rgb2gray(img);
%img = imresize(img, [150, 200]);
imshow(img);
newimg = edge(img, 'sobel', 0.1);
[length width] = size(newimg);
hold on;

interval = 0.5;
```

```
D = [];

newimg(1:15,:) = 0;
newimg(:, 1:15) = 0;
newimg(end-15:end, :) = 0;
newimg(:, end-15:end) = 0;

maxd = ceil(sqrt(length * length + width * width));

for i = -maxd:0.5:maxd
    D = [D i];
end


H = zeros(4 * maxd + 1, 91);
for i = 1:length
    for j = 1:width
        if newimg(i,j) == 1
            for theta = 2:2:182
                d = i * cosd(theta - 2) + j * sind(theta - 2);
                [d index] = min(abs(D - d));
                H(index, theta / 2) = H(index, theta / 2) + 1;
            end
        end
    end
end

R = max(H);
maximum = max(R);
threshold = 15 * double(maximum) / 100;

for i = 1:4 * maxd + 1
    for j = 2:2:182
        if H(i, j / 2) > threshold
            if i > 2 * maxd
                d = double(i - 2 * maxd - 1) / 2;
            else
                d = -1 * double(i) / 2;
            end
            th = j;
            x = 1:length;
            y = d * cscd(th - 2) - x * cotd(th - 2);
            plot(y, x, 'r');
        end
    end
end



hold off;
```

Listing 12: MATLAB source for Hough Transform of Master's commons images

```
function [ corners ] = CornerDetect(Image, nCorners, smoothSTD, windowSize)
% Function to detect corners in an image

%% Read image size
[h, w] = size(Image);

%% Smoothing with gaussian kernel
gaussian = fspecial('gaussian', windowSize, smoothSTD);
smooth_im = conv2(double(Image), gaussian, 'same');
plain = ones(windowSize);
```

```matlab

%% Calculating gradient images
[gx, gy] = gradient(smooth_im);

%% Calculating product of derivatives
Ixx = gx.*gx;
Iyy = gy.*gy;
Ixy = gx.*gy;

%% Calculating weighted sum of product of derivatives
Sxx = conv2(Ixx, plain, 'same');
Syy = conv2(Iyy, plain, 'same');
Sxy = conv2(Ixy, plain, 'same');

%% Calculating cornerness at each point
r = zeros([h,w]);
k = 0.04;
for i=25:h-25
    for j=25:w-25
        c = [Sxx(i,j), Sxy(i,j); Sxy(i,j), Syy(i,j)];
        [~,d,~] = svd(c);
        %e = eig(c);
        r(i,j) = det(d)-trace(d);
        if(r(i,j)<k)
            r(i,j) = 0;
        end
    end
end

%% Non maximal supression
ws = floor(windowSize/2);
for i=ws+1:h-ws
    for j=ws+1:w-ws
        if(r(i,j)<max(max(r(i-ws:i+ws,j-ws:j+ws))))
            r(i,j) = 0;
        end
    end
end

%% Getting nCorners best corners
[~, indices] = sort(r(:), 'descend');
[i, j] = ind2sub(size(r), indices);
corners = [i(1:nCorners),j(1:nCorners)];
end
```

Listing 13: MATLAB source for Corner Detection

Submitted by Gupta, Ajitesh
Choi, Jean
Mavalankar, Aditi Ashutosh on November 23, 2016.