CSE 252A Homework 3

Dominique Meyer A99079487, Hao-en Sung A53204772, Tsung-han Lee A53219632

November 24, 2016

1 Image warping and merging [10 pts]

Introduction

In this problem, we consider a vision application in which components of the scene are replaced by components from another image scene.

Optical character recognition, OCR, is one computer visions more successful applications. However OCR can struggle with text that is distorted by imaging conditions. In order to help improve OCR, some people will 'rectify' the image. An example is shown in Fig. 1. Reading signs from Google street view images can also benefit from techniques such as this.

This kind of rectification can be accomplished by finding a mapping from points on one plane to points another plane. In Fig 1, P_1, P_2, P_3, P_4 are mapped to $(0,0),(1,0),(1,1),(0,1)$. To solve this section of the homework, you will begin by deriving the transformation that maps one image onto another in the planar scene case. Then you will write a program that implements this transformation and uses it to rectify ads from a stadium. As a reference, see pages 316-318 in Introductory Techniques for 3-D Computer Vision by Trucco and Verri[1]

To begin, we consider the projection of planes in images. Imagine two cameras C_1 and C_2 looking at a plane π in the world. Consider a point P on the plane π and its projections $p = (u_1, v_1, 1)^\top$ in image 1 and $q = (u_2, v_2, 1)^\top$ in image 2.

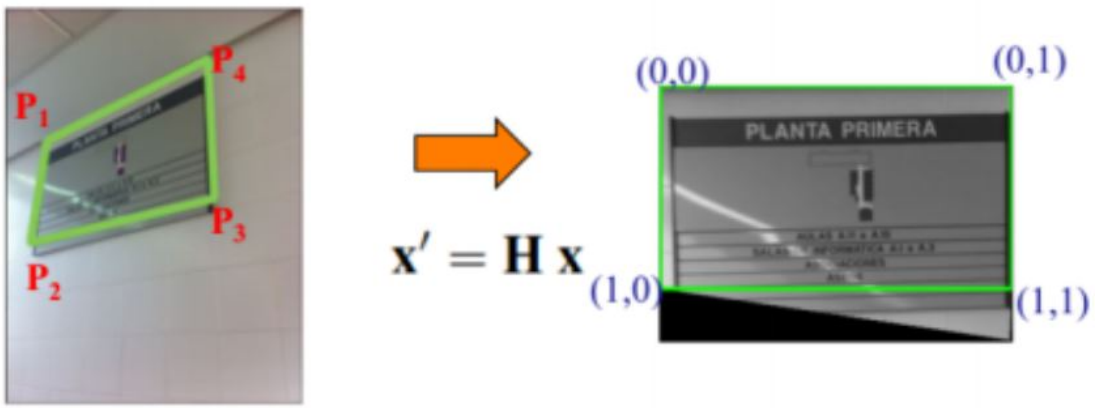

Figure 1: Input image (left) and target (right) for image mapping problem

Fact 1 *There exists a unique (up to scale) 3x3 matrix H such that, for any point P:*

$$q \equiv Hp$$

(Here $\equiv$ denotes equality in homogeneous coordinates, where the homogeneous coordinates q and p are equal up to scale) Note that H only depends on the plane and the projection matrices of the two cameras.

The interesting thing about this result is that by using H we can compute the image of P that would be seen in camera C_2 from the image of the point in camera C_1 without knowing its three-dimensional location. Such an H is a projective transformation of the plane, also referred to as a homography.

Problem definition

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.

Sol. To begin, I derived the way to solve for the Homography matrix from a set of corresponding points between both images. We begin with a set of 4 pairs of points which are obtained by clicking on the image, and the destination points which are chosen arbitrarily as the corners of a rectangle of a dimension which we decide: (p1,p′1),(p2,p′2),(p3,p′3),(p4,p′4).

The homography matrix H satisfies the equation $P' = HP$ which means that any point in the P' projection can be directly derived by multiplying the point in the P projection with the Homography matrix. The points can be combined into the matrix P and solved $PH = 0$. We can derive the Homographhraphy matrix by solving this system of equations.

$$PH = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x_1' & y_1x_1' & x_1' \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y_1' & y_1y_1' & y_1' \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2x_2' & y_2x_2' & x_2' \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2y_2' & y_2y_2' & y_2' \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3x_3' & y_3x_3' & x_3' \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3y_3' & y_3y_3' & y_3' \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4x_4' & y_4x_4' & x_4' \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4y_4' & y_4y_4' & y_4' \end{bmatrix} \begin{bmatrix} h1 \\ h2 \\ h3 \\ h4 \\ h5 \\ h6 \\ h7 \\ h8 \end{bmatrix} = 0$$

For the banners, the original points were selected by the user defining the bounding corners of the banner, and the new points were chosen as the corners of a rectangle of dimensions: 50 by 200. Having the user select the original corners and matching those to the corners of this newly defined rectangle allows us to have sufficient information to solve for the homography matrix. I decided to standardize on the point selection in a clockwise direction starting in the top left corner.

From the original and new points, the Homography matrix was then derived. In computeH.m, I created a matrix like the one stated analytically previously, and solved for the system of equation. Finally, I just had to remap the column vector to a 3x3 matrix, H.

For the warping, I began by converting the image values to double values. I then warped the corners of the selected original points to the new image space, and iterated through all

enclosed pixel values using the ndgrid function. For each value, I multiplied it by the inverse matrix H, and did a bilinear interpolation for all three color values. The result was reconverted to uint8 values, and shown with the original image which is overlayed a box of the original points. The new image did not contain a bounding box as the extremities are the bounding points: (0,0),(0,50),(200,0),(200,50).

The results can be seen here:

Figure 1: Rectification of 3 banners from the stadium image

□

2 Optical Flow [21 pts]

2.1 Dense Optical Flow [8pts]

Sol. Just as mentioned in the lectures, I first need to calculate the image gradient I_x and I_y. Later, I need to calculate the gradient of two consecutive images, i.e. I_t. After that, I can

solve a linear algebra problem

$$\begin{bmatrix} u \\ v \end{bmatrix} = (A^T A)^{-1} A^T b = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum y^2 \end{bmatrix}^{-1} \cdot \begin{bmatrix} -\sum I_x I_t \\ -\sum I_y I_t \end{bmatrix}$$

, and find out optical flow vector u and v. It is noticeable that matrix $A^T A$ must have rank two in terms of threshold τ, since other situations indicate either a flat plain or edge.

The code fulfills optical flow algorithm for *Corridor*, *Sphere* and *Synthetic Image* are included as Code 4 in Appendix; while images are shown as Fig. 2, Fig. 3, and Fig. 4. From my generated results, it can be found that algorithm with smaller window size usually has more robust performance and too large window size might lead to inaccurate flow detection.

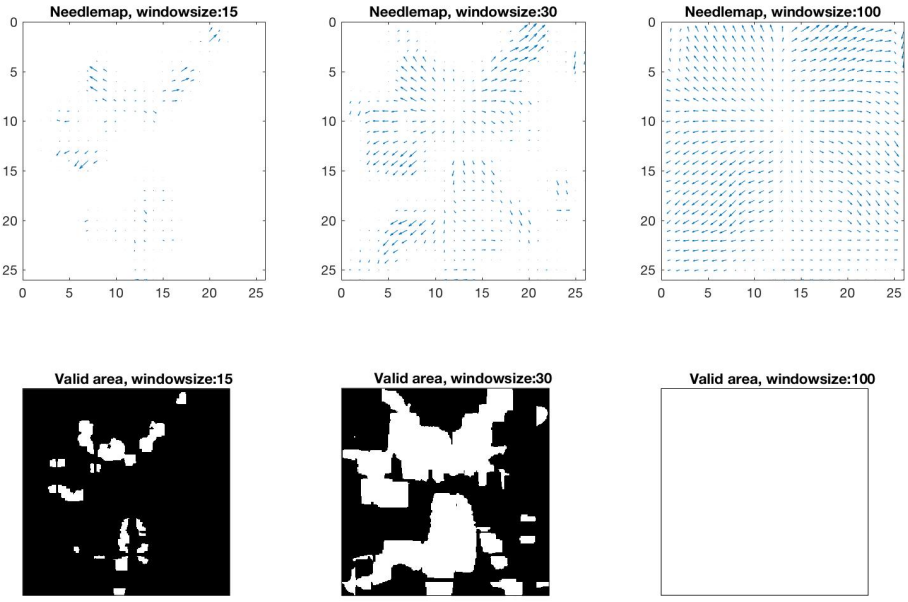

Figure 2: Dense Optical Flow: Corridor

□

2.2 Corner Detection [5pts]

Sol. Since another teammate implemented corner detection algorithm in last homework assignment, I re-implemented it again by myself this time. Basically, I calculate the image gradient I_x and I_y first then apply the non-maximum suppression algorithm to eliminate serial candidates in a local area. Later, I select top 50 candidates with largest minimum eigenvalues. The corner detection code, which is appended as Code 5 can be found in Appendix; whereas, the result images are shown as the left subfigures in Fig. 5, Fig. 6, and Fig. 7.

□

2.3 Sparse Optical Flow [8pts]

Sol. With two implemented algorithms as included in Code 4 and Code 5, I can easily make use of them to generate the right subfigures in Fig. 5, Fig. 6, and Fig. 7. My MATLAB script to generate all corresponding figures are included as Code 6.

In my experiments, I find out that 15 window size and $\tau = 0.1$ provides acceptable results for both three images. On the other hand, since focus of expansion (FOE) is defined as the intersection point of all optical flow vectors, it is clear that FOE does not exist in *Sphere* and

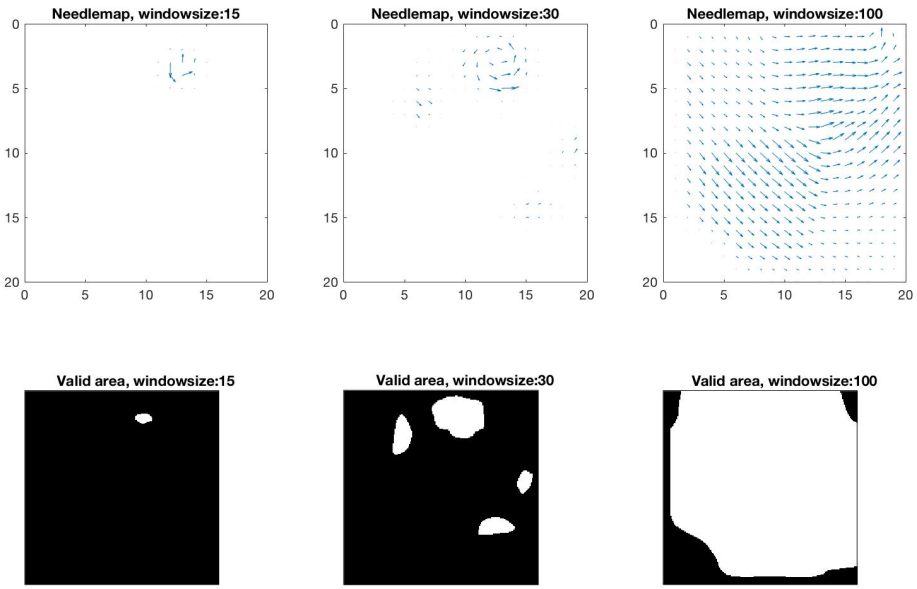

Figure 3: Dense Optical Flow: Sphere

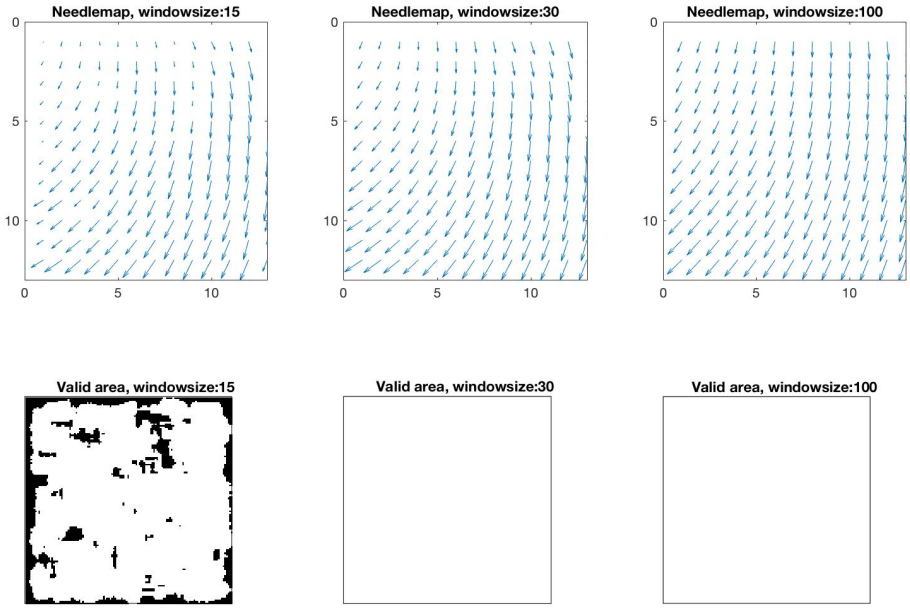

Figure 4: Dense Optical Flow: Synthetic Image

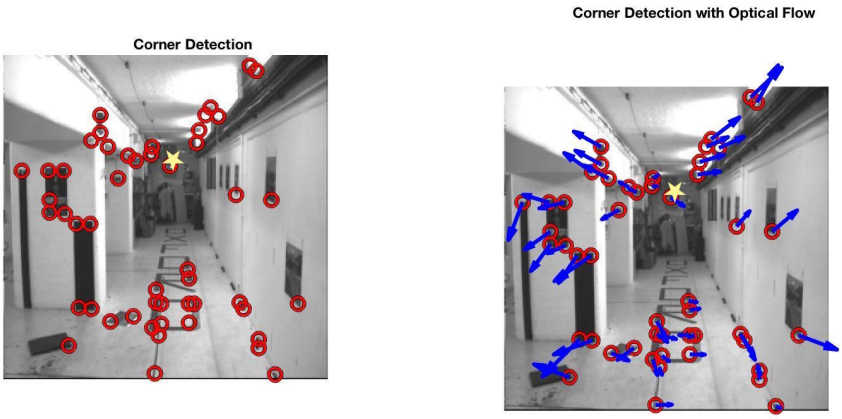

Figure 5: Sparse Optical Flow: Corridor

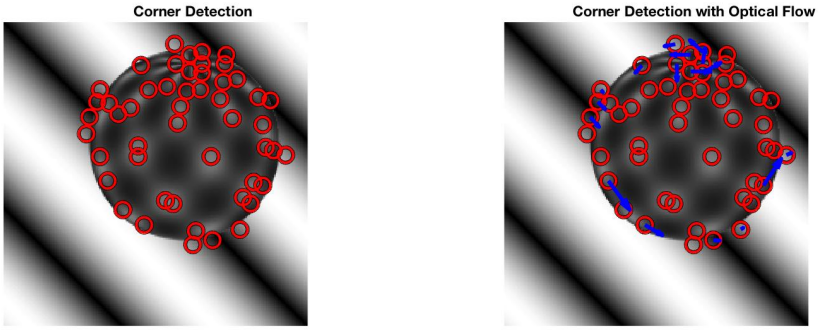

Figure 6: Sparse Optical Flow: Sphere

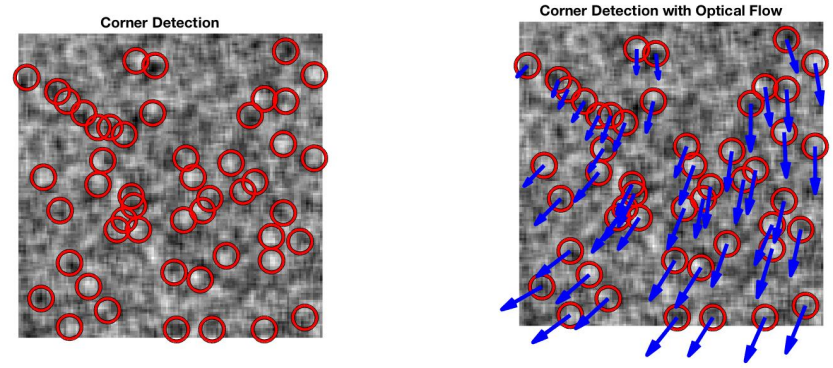

Figure 7: Sparse Optical Flow: Synthetic Image

Synthetic Image. It is clear that in these two figures, the optical vectors are surrounding but not diverting from an unknown center. Thus, I only point out the FOE for *Corridor* with a yellow star in Fig. 5. □

3 Iterative Coarse to Fine Optical Flow [15 pts]

Sol. We first determined the numbers of level of the coarse-to-fine image pyramid given the windows size, i.e. $nLevels = max(1, floor(log2(Min\,Dimension\,of\,the\,Image)+1))$. Second, we discard the first 4 levels of the pyramid, since the extreme small pixels would be the same (for example, pixels = 1), which made no improvement. It is reasonable to start from the medium-size image level. For instance, the original image is a 120×175 matrix. After setting the windows size to 15 pixels, the start level would be 15×22 matrix, followed by 30×44 matrix, 60×88 matrix, and 120×175 matrix.

Dense optical flow struggled in the case that the window size is small, since the dense optical flow algorithm assumes that the movement in the image is pretty small; whereas the coarse-to-fine algorithm is designed to overcome this limitation. When the object lies in the shallow depth, the movement of the object is larger than that in the deep depth. Thus, when the window size is small, the small movement assumption might not hold. Furthermore, when the window size is small, the two windows in the two image seems to be unmoved because no border pixel has been included. That may explain why the performance of the two algorithm deteriorate in the small window size cases. Based on the experimental result, the coarse-to-fine algorithm seems to be have better performance than dense optical flow.

The upper-level-pyramid image is shown in 8. The result images are shown as the window size equals to 5 pixels in Fig. 9, the window size equals to 10 pixels in Fig. 10, and the window size equals to 15 pixels Fig. 11 ; whereas, the Code is appended in 7, which can be found in Appendix.

□

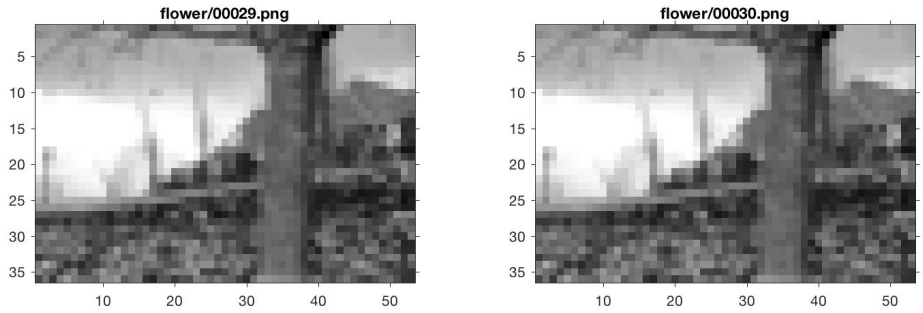

Figure 8: Image with resize (upper level of pyramid)

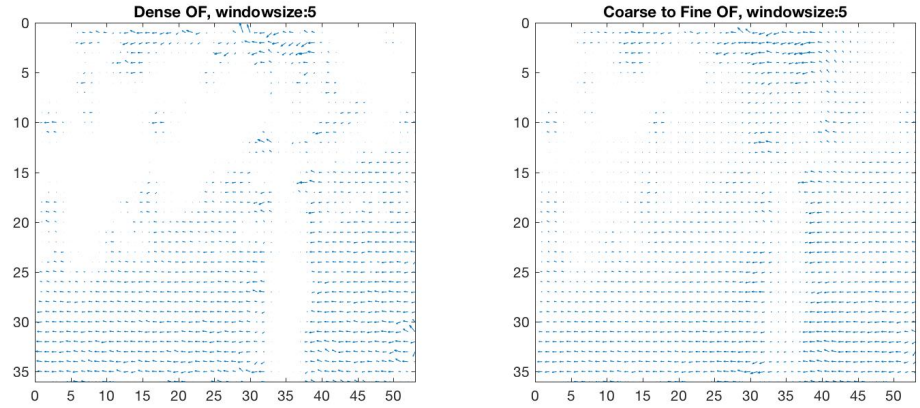

Figure 9: Optical flow with window size = 5 pixels

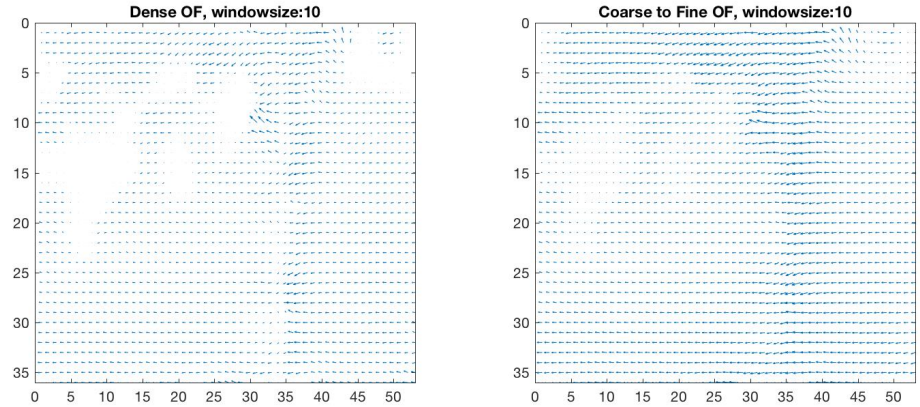

Figure 10: Optical flow with window size = 10 pixels

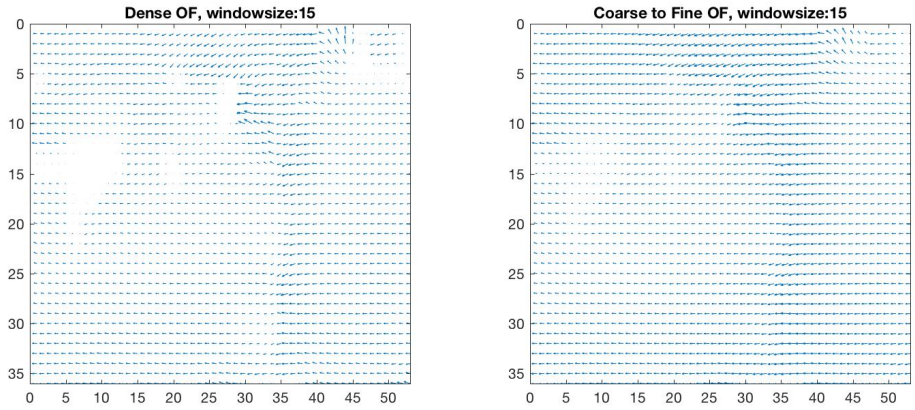

Figure 11: Optical flow with window size = 15 pixels

4 Background Subtraction and Motion Segmentation

4.1 Background subtraction [3pt]

Sol. As mentioned in the lecture, we set $I(x, y, t)$ as the foreground Image, and $I(x, y, t-1)$ as the background Image for each pair of image at certain time t and $t-1$. The difference between above two images will be something that is moving. However, even when there's nothing moving, the intensity in the image may varies over time (maybe caused by natural phenomenon or camera noise). Thus, I add a threshold $\tau = 0.5$ to filter out the natural difference, rather than that caused by movement of the objects. The formula is written as

$$Difference = |I(x, y, t) - I(x, y, t-1)| > \tau$$

If the $difference$ is larger than τ, I set the gray-scale value of the foreground image of that pixel equals to zero, i.e. the black color. After that, by calculating the mean of every processed foreground images, we have a global background for the sequence. The result image are shown as the sequence of highway in Fig. 12, and truck in Fig. 13 ; whereas, the Code is appended in 8, which can be found in Appendix.

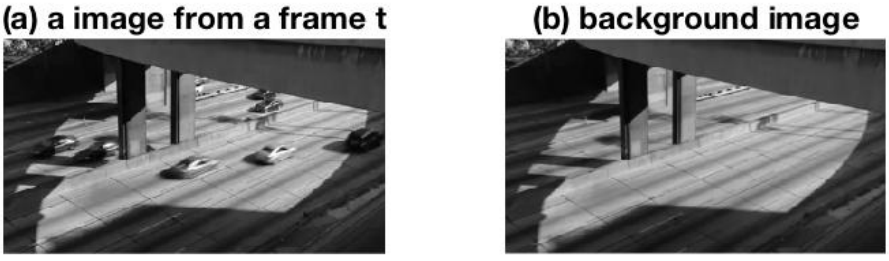

Figure 12: highway

□

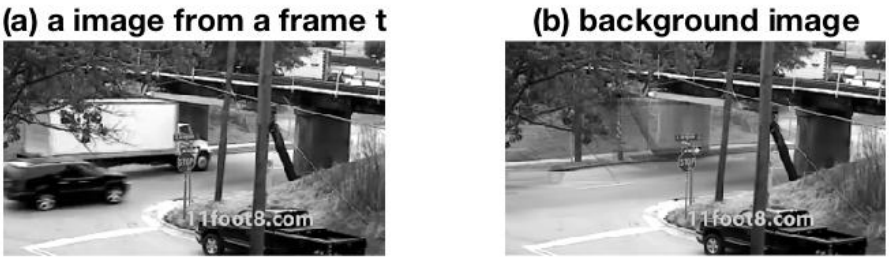

Figure 13: truck

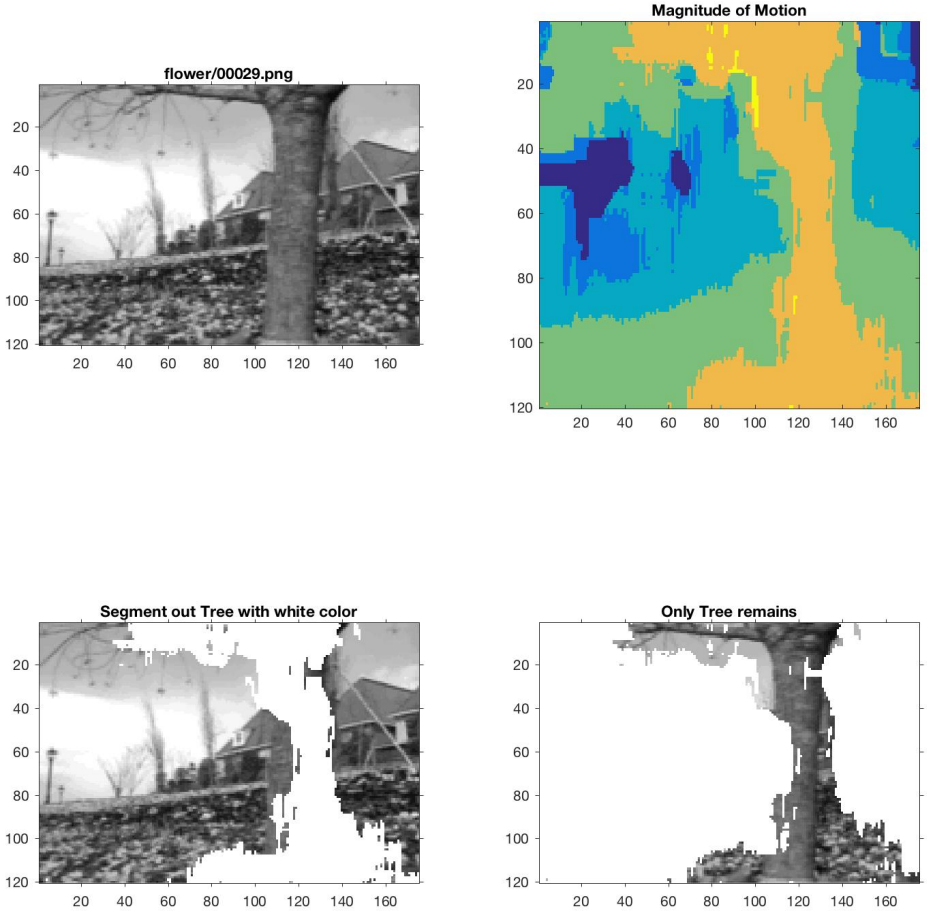

Figure 14: Motion Segmentation

4.2 Motion Segmentation [5pt]

Sol. Using the outputs of the iterative coarse-to-fine optical flow algorithm, we had two velocity matrix, i.e. u and v. The magnitude of the motion vectors at pixel (x, y) is stated as

$$Magnitude = \sqrt{u(x,y)^2 + v(x,y)^2}$$

The magnitudes of the motion vectors at different depths can be different. Specifically, the magnitudes of the motion can be larger at shallow depth than that at deep depth. Just like when you move slowly and look at the beautiful scenery, the farest mountain seems don't move at all. However, the nearest thing, like trees, moves very fast as you move.

By exploiting the phenomenon above, we can separate the magnitude of the motion into certain intervals. If the magnitude of the motion is between 2.6 and 3.8, then masking out the segmented-out-tree image with white color; whereas, the other only-tree-in image contains the pixel from the original image. If the magnitude of the motion doesn't lie in the interval above, then do the opposite action.

The result image are shown in Fig. 14; whereas, the Code is appended in 11, which can be found in Appendix.

□

5 Hough Lines [10 Pts]

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 as shown in figure 9. Compute and display its Hough Transform; the result should look like figure 9. 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.
- 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% maximum accumulator count over the entire HT, i.e. $0.75 \times max(HT(:))$.
- Now that you have mastered Hough transform, Repeat the procedure for MS Commons room images ($common1.jpg$ & $common2.jpg$). Set the appropriate threshold parameters to plot corresponding lines atop the original image. You will be graded on the accuracy of the result.

Sol. The first part of this problem involves the creation of a test image, and deriving the Hough Transform of it. With the Hough Transform, we can set a threshold at which a peak is created, and when re projected into the image space, this becomes a line. There are 3 parameters which can be change, theta resolution, delta resolution and the threshold. Adjusting all of these allows us to get an ideal balance of number of lines. A matrix is created which contains the vote count and which then iterates through all pixel values for potential points. Only the points above the threshold are kept. For the first part, this threshold is 2 votes per hough segment. For the second part, this is 0.75 and for the others it is between 0.7 and 0.8. The parameters theta and delta are varied for optimal results.

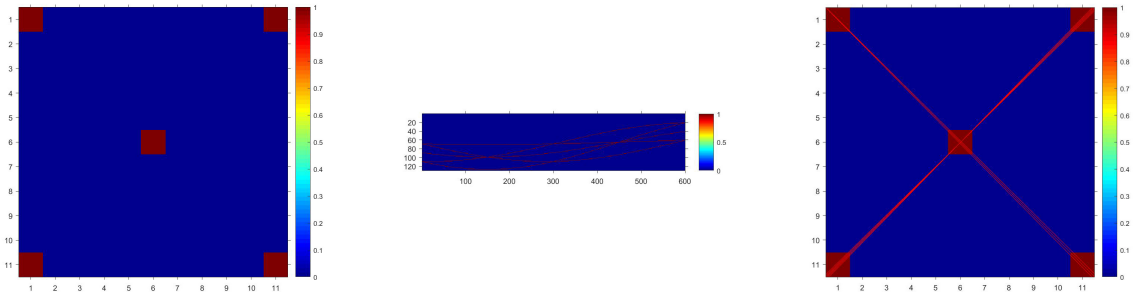

Figure 15: Original test image, Hough Transformation, Hough Lines

For the lanes and the commons images, both came out the best using the Sobel operator even though I tried with canny edge detection. The parameters are labeled in the images. For the theta resolution 0.3 was used and for the delta resolution, 0.25. This was for the lanes and commons images. The results suggest successful edge detection using hough transforms.

Figure 16: Rectification of 3 banners from the stadium image

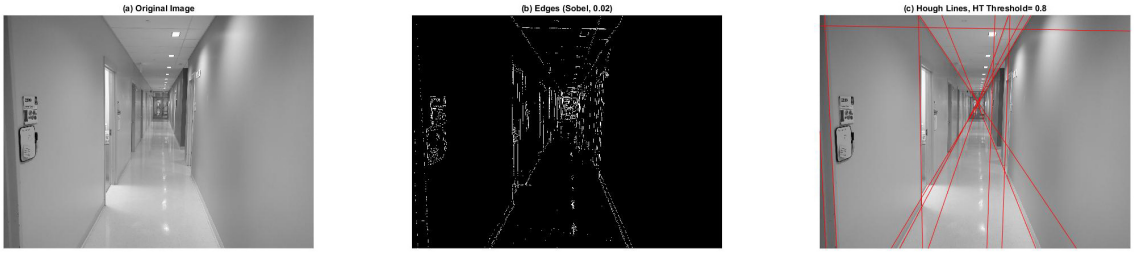

Figure 17: Rectification of 3 banners from the stadium image

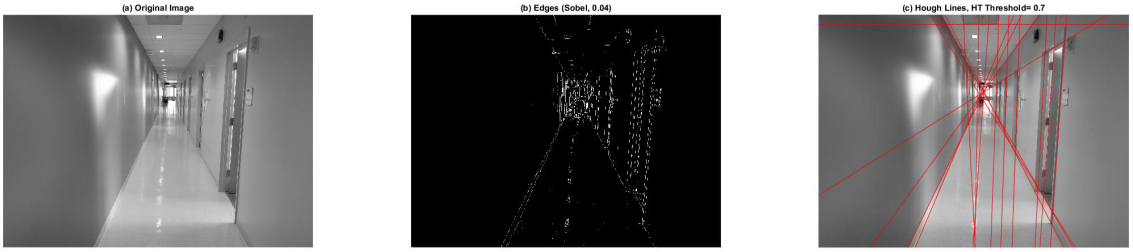

Figure 18: Rectification of 3 banners from the stadium image

□

6 Appendix

```
%% Reads image file and allows user to select add to reproject
I1=imread('stadium.jpg');
%select points on the image, preferable the corners of an ad.
imshow(I1)
points = ginput(4);
figure(1)
subplot(1,2,1);
imshow(I1);
new_points = [0,0; 200,0; 200,50; 0,50]; %creates new image of specified size
H= computeH(points,new_points);
%warp will return just the ad rectified
warped_img = warp(I1,new_points, H);
figure(2); imshow(warped_img);
figure(1); plotsquare([points';1 1 1 1]); imshow(I1)
```

Listing 1: Problem 1: Running Script

```
%% Matlab Script that computes Homography matrix from 4 pairs of points

function H = computeH(points, new_points)

  n = size(points, 1);
  A = zeros(n*3,9);
  b = zeros(n*3,1);
for i=1:n
    A(3*(i-1)+1,1:3) = [new_points(i,:),1];
    A(3*(i-1)+2,4:6) = [new_points(i,:),1];
    A(3*(i-1)+3,7:9) = [new_points(i,:),1];
    b(3*(i-1)+1:3*(i-1)+3) = [points(i,:),1];
end
x = (A\b)';
H = [x(1:3); x(4:6); x(7:9)];

end
```

Listing 2: Problem 1: Compute Homography Matrix

```
%% Warp script which reprojects image to desired rectangle

function warped_img = warp(I1, new_points, H)

I1=im2double(I1);
[h w d] = size(I1);

% find and warp incoming corners to new image space
cp = H*[ 1 1 w w ; 1 h 1 h ; 1 1 1 1 ];
Xpr = min( cp(1,:) ) : max( cp(1,:) ); % x range
Ypr = min( cp(2,:) ) : max( cp(2,:) ); % y range

[Xp,Yp] = ndgrid(Xpr,Ypr);
[wp hp] = size(Xp);
% inverse transform with Homography Matrix
n = wp*hp;
X = H \ [ Xp(:) Yp(:) ones(n,1) ]';  % warp
% bilinear interpolation
xI = reshape( X(1,:),wp,hp)';
yI = reshape( X(2,:),wp,hp)';
```

```
Ip(:,:,1) = interp2(I1(:,:,1), xI, yI, '*bilinear'); % red
Ip(:,:,2) = interp2(I1(:,:,2), xI, yI, '*bilinear'); % green
Ip(:,:,3) = interp2(I1(:,:,3), xI, yI, '*bilinear'); % blue

warped_img=im2uint8(Ip);

end
```

Listing 3: Problem 1: Warp Image

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

%% Parameters
[n, m] = size(I1);
hw = fix(windowSize/2);

%% Calculate Gradient
[Gx, Gy] = gradient(I1);
Gt = I2 - I1;

%% Precalculate Squared Gradient
Gxx = Gx .* Gx;
Gxy = Gx .* Gy;
Gyy = Gy .* Gy;
Gxt = Gx .* Gt;
Gyt = Gy .* Gt;

%% Optical Flow
u = zeros(n, m);
v = zeros(n, m);
hitMap = false(n, m);
for r = 1:n
    for c = 1:m
        vxx = sum(sum(Gxx(max(r-hw, 1):min(r+hw,n), max(c-hw, 1):min(c+hw,m))));
        vyy = sum(sum(Gyy(max(r-hw, 1):min(r+hw,n), max(c-hw, 1):min(c+hw,m))));
        vxy = sum(sum(Gxy(max(r-hw, 1):min(r+hw,n), max(c-hw, 1):min(c+hw,m))));
        vxt = sum(sum(Gxt(max(r-hw, 1):min(r+hw,n), max(c-hw, 1):min(c+hw,m))));
        vyt = sum(sum(Gyt(max(r-hw, 1):min(r+hw,n), max(c-hw, 1):min(c+hw,m))));

        ATA = [vxx, vxy; vxy, vyy];
        ATb = [-vxt; -vyt];

        if min(eig(ATA)) > tau
            V = ATA \ ATb;
            u(r, c) = V(1);
            v(r, c) = V(2);
            hitMap(r, c) = 1;
        end
    end
end
```

Listing 4: Problem 2-1: Optical Flow

```
function corners = cornerDetect(I)

%% Parameters
windowSize = 7;
smoothSTD = 1;
num = 50;
```

```
[n, m] = size(I);
hw = fix(windowSize/2);

%% Gaussian Smoothing
H = fspecial('gaussian', [5, 5], smoothSTD);
I = imfilter(I, H, 'replicate');

%% Calculate Gradient
[Gx, Gy] = gradient(I);

%% Precalculate Squared Gradient
Gxx = Gx .* Gx;
Gxy = Gx .* Gy;
Gyy = Gy .* Gy;

%% Corner Detection
em = zeros(n, m);
mat = zeros(n*m, 3);
for r = 1:n
    for c = 1:m
        vxx = sum(sum(Gxx(max(r-hw, 1):min(r+hw,n), max(c-hw, 1):min(c+hw,m))));
        vyy = sum(sum(Gyy(max(r-hw, 1):min(r+hw,n), max(c-hw, 1):min(c+hw,m))));
        vxy = sum(sum(Gxy(max(r-hw, 1):min(r+hw,n), max(c-hw, 1):min(c+hw,m))));
        ATA = [vxx, vxy; vxy, vyy];
        em(r, c) = min(eig(ATA));
    end
end

%% Non-maximum Supression
for r = 1:n
    for c = 1:m
        vmax = max(max(em(max(r-hw, 1):min(r+hw,n), max(c-hw, 1):min(c+hw,m))));
        if em(r, c) == vmax
            mat(r*n+c, :) = [c, r, em(r, c)];
        end
    end
end

%% Return Top-k Corners
[~, idx] = sort(mat(:,3), 'descend');
mat = mat(idx, :);
corners = mat(1:num, 1:2);
```

Listing 5: Problem 2-2: Corner Detection

```
%% Read files
crf = imread('../dat/corridor/bt.000.png');
crf = mat2gray(crf);
crs = imread('../dat/corridor/bt.001.png');
crs = mat2gray(crs);

spf = imread('../dat/sphere/sphere.0.png');
spf = rgb2gray(spf);
spf = mat2gray(spf);
sps = imread('../dat/sphere/sphere.1.png');
sps = rgb2gray(sps);
sps = mat2gray(sps);

syf = imread('../dat/synth/synth_000.png');
```

```
syf = mat2gray(syf);
sys = imread('../dat/synth/synth_001.png');
sys = mat2gray(sys);

%% Prepare dataset
windowSizeList = [15, 30, 100];
dataList = {crf, crs; spf, sps; syf, sys};
dataName = {'corridor', 'sphere', 'synth'};

%% Problem 2-1
for i = 1:3 % three dataset
    res = figure('visible', 'off');
    set(res, 'PaperPosition', [0 0 12 8]);
    data = dataList(i, :);
    for j = 1:3 % three windowSize
        windowSize = windowSizeList(j);
        [u, v, hitMap] = opticalFlow(data{1}, data{2}, windowSize, 0.5);
        nu = imresize(u, 0.1);
        nv = imresize(v, 0.1);
        [n, m] = size(nu);
        [X, Y] = meshgrid(1:m, 1:n);

        sp = subplot(2,3,j);
        quiver(X, Y, nu, nv);
        set(sp, 'XLim', [0, m], 'XTick', 0:5:m);
        set(sp, 'YLim', [0, n], 'YTick', 0:5:n);
        set(sp, 'ydir', 'reverse');
        title(strcat('Needlemap, windowsize: ', int2str(windowSize)));

        subplot(2,3,3+j);
        hold on
        imshow(hitMap);
        [n, m] = size(u);
        rectangle('Position',[0, 0, n, m],'EdgeColor','k');
        hold off
        title(strcat('Valid area, windowsize: ', int2str(windowSize)));
    end

    saveas(res, strcat('../res/denseOpticalFlow_', dataName{i}, '.jpg'));
end

for i = 1:3 % three dataset
    res = figure('visible', 'off');
    set(res, 'PaperPosition', [0 0 12 8]);
    data = dataList(i, :);

    %% Problem 2-2
    corners = cornerDetect(data{1});

    subplot(1,2,1);
    hold on
    imshow(data{1});
    viscircles(corners, meshgrid(5, 1:50));
    hold off
    title(strcat('Corner Detection'));

    %% Problem 2-3
    vx = zeros(50, 1);
```

```
    vy = zeros(50, 1);
    vu = zeros(50, 1);
    vv = zeros(50, 1);
    [u, v, hitMap] = opticalFlow(data{1}, data{2}, 15, 0.1);
    for j = 1:50
        vx(j) = corners(j,1);
        vy(j) = corners(j,2);
        vu(j) = u(vy(j), vx(j));
        vv(j) = v(vy(j), vx(j));
    end

    subplot(1,2,2);
    hold on
    imshow(data{1});
    viscircles(corners, meshgrid(5, 1:50));
    quiver(vx, vy, vu, vv, 'color', [0, 0, 1], 'LineWidth', 3);
    hold off
    title(strcat('Corner Detection with Optical Flow'));

    saveas(res, strcat('../res/sparseOpticalFlow_', dataName{i}, '.jpg'));
end
```

Listing 6: Problem 2: Running Script

```
%% Read files
I1 = rgb2gray( imread('data/flower/00029.png') );
I1 = mat2gray(I1);

I2 = rgb2gray( imread('data/flower/00030.png') );
I2 = mat2gray(I2);


%% Coarse-to-Fine Estimation
windowSize = 15;
tau = 0.02;


[n, m] = size(I1);
MinDim = min( [n, m] );
nLevels = max(1, floor( log2( MinDim) )+1 );
startLevelofPyramid = 4;

% Pyramid Calculation
for i = startLevelofPyramid:nLevels
    reSize = 2^(nLevels - i);
    h_i = round( n/reSize );
    w_i = round( m/reSize );

    I1_i = imresize(I1,[h_i w_i], 'bilinear');
    I2_i = imresize(I2,[h_i w_i], 'bilinear');
    if (i == startLevelofPyramid)
        [u_pre, v_pre, hitMap] = opticalFlow(I1_i, I2_i, windowSize, tau);
    else
        u_pre = resizem(u_pre,[h_i w_i],'bilinear');
        v_pre = resizem(v_pre,[h_i w_i],'bilinear');
        [u_pri, v_pri, hitMap]=opticalFlow2(I1_i, I2_i, u_pre, v_pre, windowSize,
            tau);
        u_pre = u_pre + u_pri;
        v_pre = v_pre + v_pri;
```

```
    end
end
%% Result of Coarse-to-Fine optical flow
u2 = u_pre;
v2 = v_pre;

dlmwrite('Output/u2.txt', u2);
dlmwrite('Output/v2.txt', v2);


%% Result of dense optical flow
[u, v, hitMap] = opticalFlow(I1, I2, windowSize, tau);


%% Plotting
graph = figure('visible', 'off');
set(gcf,'units','points','position',[0,0,800,310]);

% Plot Dense Optical flow
sp = subplot(1,2,1);
nu = imresize(u, 0.3);
nv = imresize(v, 0.3);

[n, m] = size(nu);
[X, Y] = meshgrid(1:m, 1:n);
quiver(X, Y, nu, nv);
set(gca,'ydir','reverse');
set(sp, 'XLim', [0, m], 'XTick', 0:5:m);
set(sp, 'YLim', [0, n], 'YTick', 0:5:n);
title(strcat('Dense OF, windowsize: ', int2str(windowSize)));

% Plot Coarse to fine Optical flow
sp2 = subplot(1,2,2);
nu2 = imresize(u2, 0.3);
nv2 = imresize(v2, 0.3);

[n2, m2] = size(nu2);
[X2, Y2] = meshgrid(1:m2, 1:n2);
quiver(X2, Y2, nu2, nv2);
set(gca,'ydir','reverse');
set(sp2, 'XLim', [0, m2], 'XTick', 0:5:m2);
set(sp2, 'YLim', [0, n2], 'YTick', 0:5:n2);
title(strcat('Coarse to Fine OF, windowsize: ', int2str(windowSize)));

saveas(graph, 'Output/3-3.jpg');


%% Plotting original graph
graph = figure('visible', 'off');
set(gcf,'units','points','position',[0,0,800,400]);

subplot(1,2,1);
imshow(imresize(I1, 0.3))
axis on;
title('flower/00029.png');

subplot(1,2,2);
imshow(imresize(I1, 0.3))
```

```
axis on;
title('flower/00030.png');
saveas(graph, 'Output/3-original.jpg');


function [u, v, hitMap] = opticalFlow2(I1, I2, u_pre, v_pre, windowSize, tau)

%% Parameters
[n, m] = size(I1);
hw = fix(windowSize/2);

%% Calculate Gradient
[Gx, Gy] = gradient(I1);

%% Precalculate Squared Gradient
Gxx = Gx .* Gx;
Gxy = Gx .* Gy;
Gyy = Gy .* Gy;

%% Optical Flow
u = zeros(n, m);
v = zeros(n, m);
hitMap = false(n, m);
for r = 1:n
    for c = 1:m
        x_LB = max(r-hw, 1);
        x_HB = min(r+hw,n);
        y_LB = max(c-hw, 1);
        y_HB = min(c+hw,m);

        vxx = sum(sum(Gxx(x_LB:x_HB, y_LB:y_HB)));
        vyy = sum(sum(Gyy(x_LB:x_HB, y_LB:y_HB)));
        vxy = sum(sum(Gxy(x_LB:x_HB, y_LB:y_HB)));

        Gt = zeros(x_HB-x_LB+1, y_HB-y_LB+1);
        for i = 1:x_HB-x_LB+1
            for j = 1:y_HB-y_LB+1
                x_disp = round( i+u_pre(i,j) );
                y_disp = round( j+v_pre(i,j) );
                if(x_LB <= x_disp && x_disp <= x_HB ...
                   && y_LB <= y_disp && y_disp <= y_HB)
                    Gt(i,j) = I2(x_disp, y_disp) - I1(i,j);
                else
                    Gt(i,j) = 0;
                end
            end
        end

        Gxt = Gx(x_LB:x_HB, y_LB:y_HB) .* Gt;
        Gyt = Gy(x_LB:x_HB, y_LB:y_HB) .* Gt;

        vxt = sum(sum(Gxt));
        vyt = sum(sum(Gyt));

        ATA = [vxx, vxy; vxy, vyy];
        ATb = [-vxt; -vyt];

        if min(eig(ATA)) > tau
```

```
            V = ATA \ ATb;
            u(r, c) = V(1);
            v(r, c) = V(2);
            hitMap(r, c) = 1;
        end
    end
end
end
```

Listing 7: Problem 3: Running Script

```

tau = 25;

%framesequence = 'data/highway/';
framesequence = 'data/truck/';
Img = backgroundSubtract(framesequence, tau);

% ---- Plotting ----
graph = figure('visible', 'off');
set(gcf,'units','points','position',[0,0,400,200]);

subplot (1,2,1);
%It = im2double( imread( strcat(framesequence, '00050.png') ) );
It = im2double( imread( strcat(framesequence, '00395.png') ) );
imshow(It);
title('(a) a image from a frame t');

subplot(1,2,2);
imshow(Img);
title('(b) background image');

%saveas(graph, 'Output/4-1_SubBackground-1.jpg');
saveas(graph, 'Output/4-1_SubBackground-2.jpg');

%% Background Substruction
function Img = backgroundSubtract(framesequence, tau)
    imgSeqFiles = dir( strcat(framesequence,'*.png') );
    nFiles = length(imgSeqFiles);
    I = imread( strcat(framesequence,imgSeqFiles(1).name) );

    [nRows, nCols] = size(I);
    backImgSum = double( zeros(nRows, nCols) );

    for ii = 2:nFiles
        It_0 = im2double( imread( strcat(framesequence,imgSeqFiles(ii-1).name) ) );
        It_1 = im2double( imread( strcat(framesequence,imgSeqFiles(ii).name) ) );
        Img = imageDifference(It_0, It_1, tau);
        backImgSum = backImgSum + Img;
    end

    Img = backImgSum / (nFiles-1);

end

function SubBackImg = imageDifference(It_0, It_1, tau)
    SubBackImg = It_1 - It_0;
    [nRows, nCols] = size(SubBackImg);
    for i = 1:nRows
```

```
        for j = 1:nCols
            % If exceed difference threshold, then set the pixel value to
            % zero
            if ( abs(SubBackImg(i,j)) > tau )
                SubBackImg(i,j) = It_0(i,j) - It_1(i,j);
            else
                SubBackImg(i,j) = It_0(i,j);
            end
        end
    end

end
```

Listing 8: Problem 4: Running Script

```
u2 = dlmread('data/u2.txt');
v2 = dlmread('data/v2.txt');

It = im2double( imread( strcat('data/flower/', '00029.png') ) );
It = rgb2gray(It);

[n, m] = size(It);

It_segTree = zeros(n, m);
I_OnlyTree = zeros(n, m);
MotSegFig = zeros(n, m);
threhold = [2.6,3.8];
SegLst = [1, 1.5, 1.9, 2.6, 3.8];


for i = 1:n
    for j = 1:m
        MagOfMotion = sqrt(u2(i,j)^2 + v2(i,j)^2 );
        if (MagOfMotion <= SegLst(1))
            MotSegFig(i,j) = 1;
        elseif ( SegLst(1) < MagOfMotion && MagOfMotion <= SegLst(2) )
            MotSegFig(i,j) = 2;
        elseif ( SegLst(2) < MagOfMotion && MagOfMotion <= SegLst(3) )
            MotSegFig(i,j) = 3;
        elseif ( SegLst(3) < MagOfMotion && MagOfMotion <= SegLst(4) )
            MotSegFig(i,j) = 4;
        elseif ( SegLst(4) < MagOfMotion && MagOfMotion <= SegLst(5) )
            MotSegFig(i,j) = 5;
        else
            MotSegFig(i,j) = 6;
        end
        %MotSegFig(i,j) = MagOfMotion;

        if(threhold(1)<MagOfMotion && MagOfMotion<threhold(2))
            It_segTree(i,j) = 255;
            I_OnlyTree(i,j) = It(i,j);
        else
            It_segTree(i,j) = It(i,j);
            I_OnlyTree(i,j) = 255;
        end
    end
end

graph = figure('visible', 'on');
```

```
set(gcf,'units','points','position',[0,0,800,800]);

subplot(2,2,1);
imshow(It);
axis on;
title('flower/00029.png');

subplot(2,2,2);
%colormap('jet');
imagesc(MotSegFig);
%colorbar;
title('Magnitude of Motion');

subplot(2,2,3);
imshow(It_segTree);
axis on;
title('Segment out Tree with white color');

subplot(2,2,4);
imshow(I_OnlyTree);
axis on;
title('Only Tree remains');
saveas(graph, 'Output/4-2.jpg');
```

Listing 9: Problem 4: Running Script

```
%% ------------ Parameters ----------------
n=11; %square dimensions of test image
t_res = 0.3 ; % theta resolution
d_res =0.25 ; % delta resolution

%% ------------ Generate Test Image ----------------
img = zeros(n);
img(1,1) = 1;
img(n,1) = 1;
img(1,n) = 1;
img(n,n) = 1;
img(round(n/2),round(n/2)) = 1;

subplot(1,3,1); imshow(img,'Colormap',jet, 'InitialMagnification',4000); colorbar ;
    axis on ;hold on ;
%% ------------ Computing Hough Transform of Image ----------------
H = zeros(1,180/t_res) ;
t = atan2(size(img,1 ) ,size(img,2)) ;
rowSize = ceil(size(img,1)*sin(t)+size(img,2)*cos(t));
T =0 ; % couting variable
img_H = zeros(2*(rowSize/d_res)+1,180/t_res) ;
for row = 1:size(img,1)
    for col =1:size(img,2)
        T = T+1;
        if(img(row,col)== 1)
            for degree = t_res:t_res:180
                d =(col*cosd(degree)+row*sind(degree));
                H(T,round(degree/t_res)) = d ;
                img_H(round(d/d_res) + rowSize/d_res+1,round(degree/t_res))=img_H(
                    round(d/d_res)+rowSize/d_res+1, round(degree/t_res))+1;
            end
        end
    end
```

```
end
subplot(1,3,3); imshow(img,'Colormap',jet, 'InitialMagnification',6000); colorbar ;
    axis on ;hold on ;

%% ---------------- Threshold peak detection and Hough line generation
    ---------------------
clear row col d x y
T = 0 ;
for row =1:size(img_H,1)
    for col= 1:size(img_H,2)
        if(img_H(row,col) > 2)
            T = T +1;
            d =(row-1-rowSize /d_res)*d_res ;
            t = col*t_res ;
            fy =@(x) (d-x*cosd(t))/sind(t) ;
            fplot(fy,'Color', 'r') ;
        end
    end
end
subplot(1,3,2); imshow(img_H,'Colormap',jet,'InitialMagnification',4000); colorbar
    ;axis on ;hold on ;
```

Listing 10: Problem 5: Part 1 Hough Trans and Lines for test image

```
%% ------------- Parameters ----------------
t_res = 0.3 ; % theta resolution
d_res =0.25 ; % delta resolution
threshold =0.75; % threshold value for hough peaks

%% ------------- Importing Image and converting to Grayscale -------------
I = imread('lanes.png');
I = rgb2gray(I);
subplot(1,3,1);imshow(I); title('(a) Original Image'); hold on ;

%% ------------- Using Edge Detector -------------
E = edge(I, 'sobel',0.05);
subplot(1,3,2); imshow(E); title('(b) Edges (Sobel, 0.05)'); hold on ;

%% ------------- Hough Transform of Image -------------
%[H, theta, rho] = hough(E); -- inbuilt Matlab function to check result

H = zeros(1,180/t_res) ;
t = atan2(size(E,1 ) ,size(E,2)) ;
rowSize = ceil(size(E,1)*sin(t)+size(E,2)*cos(t));
T =0 ; % couting variable
img_H = zeros(2*(rowSize/d_res)+1,180/t_res) ;
for row = 1:size(E,1)
    for col =1:size(E,2)
        T = T+1;
        if(E(row,col)== 1)
            for degree = t_res:t_res:180
                d =(col*cosd(degree)+row*sind(degree));
                H(T,round(degree/t_res)) = d ;
                img_H(round(d/d_res) + rowSize/d_res+1,round(degree/t_res))=img_H(
                    round(d/d_res)+rowSize/d_res+1, round(degree/t_res))+1;
            end
        end
    end
end
```

```
%% ---------------- Threshold peak detection and Hough line generation
    ---------------------
clear row col d x y
T = 0 ;
max_img = max(max(img_H));
for row =1:size(img_H,1)
    for col= 1:size(img_H,2)
        if(img_H(row,col) > threshold*max_img)
            T = T +1;
            d =(row-1-rowSize /d_res)*d_res ;
            t = col*t_res ;
            fy =@(x) (d-x*cosd(t))/sind(t) ;
            fplot(fy,'Color', 'r') ;
        end
    end
end
subplot(1,3,3); imshow(I); title('(d) Hough Lines on Original Image'); hold on ;
```

Listing 11: Problem 5: Part 2 Hough Trans and Lines for general images