LABORATORY #6: REPORT
A.A. 2015-2016

Roberto Costa

June 1, 2016

Subject:	Image and Video Analysis
Delivery date:	31-05-2016
Professor:	P. Zanuttigh
Student number:	1128285

1 Motion detection

The purpose of the lab is to develop an algorithm to detect the motion on some video.
The problem to estimate the motion on a video can be solved by estimating the motion between two adjacent frames of the video. The problem can be solved by estimating the SIFT feature between the two images and by finding the corresponding vector for each SIFT feature. The approach just descripted is computationally expensive and it is not used on some real time applications.
The algorithm is developed under the assumption of color constancy between two images, the assumption of small motion and the assumption of $I(x, y, t)$ differentiable.
The first assumptions can be formalized by the folowing equation

$$I\left(x\left(t+\Delta t\right), y\left(t+\Delta t\right), t+\Delta t\right) - I\left(x\left(t\right), x\left(t\right), t\right) = 0 \tag{1}$$

By Taylor approximation of the first term of the sum, we can write

$$I\left(x\left(t+\Delta t\right), y\left(t+\Delta t\right), t+\Delta t\right) \sim I\left(x\left(t\right), y\left(t\right), t\right) + \frac{\partial I}{\partial x}\frac{\partial x}{\partial t}\Delta t + \frac{\partial I}{\partial y}\frac{\partial y}{\partial t}\Delta t + \frac{\partial I}{\partial t}\frac{\partial t}{\partial t}\Delta t \tag{2}$$

By combining eq. (1) and (2) we get

$$\frac{\partial I}{\partial x}v_x + \frac{\partial I}{\partial y}v_y + \frac{\partial I}{\partial t} = 0 \tag{3}$$

which can be written as

$$\left[\Delta I(p_i)\right]^T \begin{bmatrix} v_x \\ v_y \end{bmatrix} + I_t(p_i) = 0 \tag{4}$$

For each pixel, the eq. (4) cannot be solved because the constraints are not enough with respect to the unknown variables.

This problem can be solved by imposing additional constraints, which pratically means to evaluate the equation for a window of pixels in the neighborhood:

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \cdots & \cdots \\ I_x(p_{N^2}) & I_y(p_{N^2}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \cdots \\ I_t(p_{N^2}) \end{bmatrix} \tag{5}$$

where a squared window of size N is used.
The target to minimize the first term of eq. (3) can be achieved by minimizing the norm of it. By calling the first matrix of the first term of eq. (5) "A", the second matrix of the first term d and the second term b, we can write $Ad = b$. By pre-multiplicating both terms of eq. (5) by A^T we get

$$A^T Ad = -A^T b \tag{6}$$

which is a system of 2 equations in 2 unknown which can be solved if matrix $A^T A$ is invertible.
Some other constraints for the solvability of eq. (5) are that matrix $A^T A$ should not be too small, otherwise the noise dominates, and that matrix $A^T A$ should be well conditioned.
In terms of eigenvalues, λ_1 and λ_2, they need to be not to small, to avoid noise domination, and the ratio between the bigger and the lower λ_1/λ_2 should be not to large.
Since matrix $A^T A$

$$A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} = \sum \Delta I \, (\Delta I)^T \tag{7}$$

is the second moment matrix, we can recall Harris corner detector to find corners, edge region and flat region; in particular we have a corner if both eigenvalues of $M = A^T A$ are large, we have an edge if only one of them is large and we have a flat region if both eigenvalues are small.
The best keypoints for motion detection are the corners.
By increasing the window size we can notice that the motion estimation is more accurate, even if the number of keypoints is lower. Some frames with different video input and with different window size are reported below.

2 Ouput figures

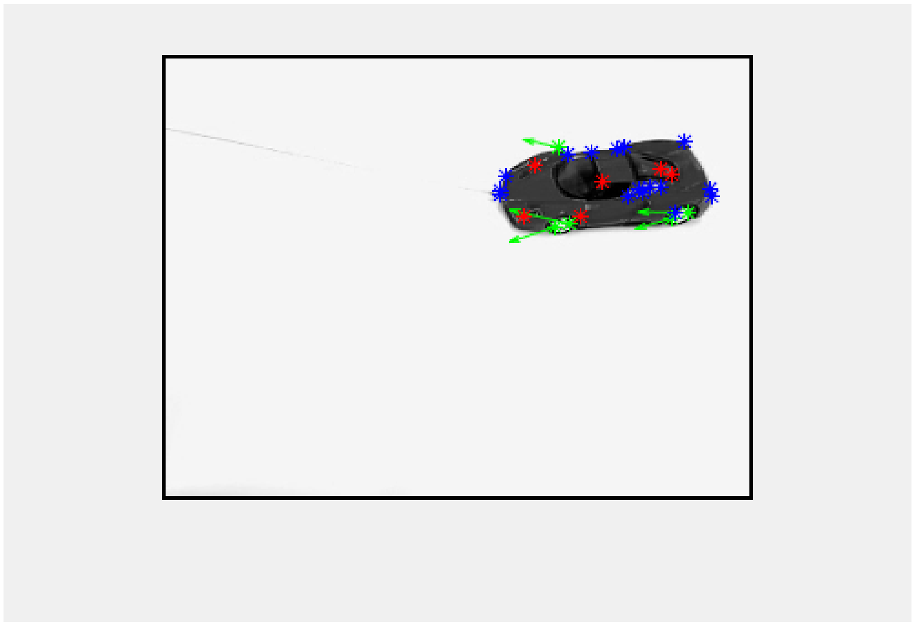

Figure 1: Output video 1, frame # 66, window size: 2x2 pixels

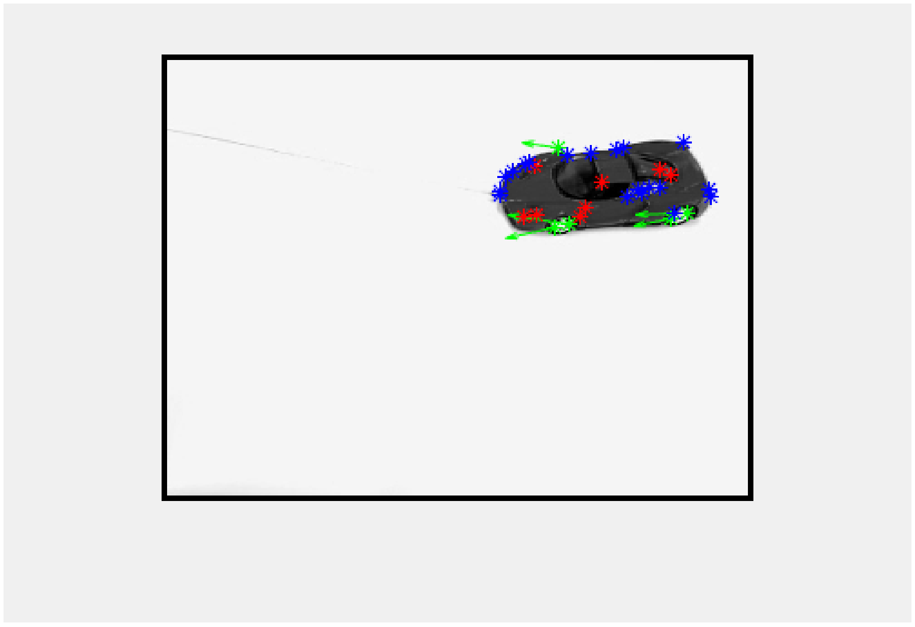

Figure 2: Output video 1, frame # 66, window size: 3x3 pixels

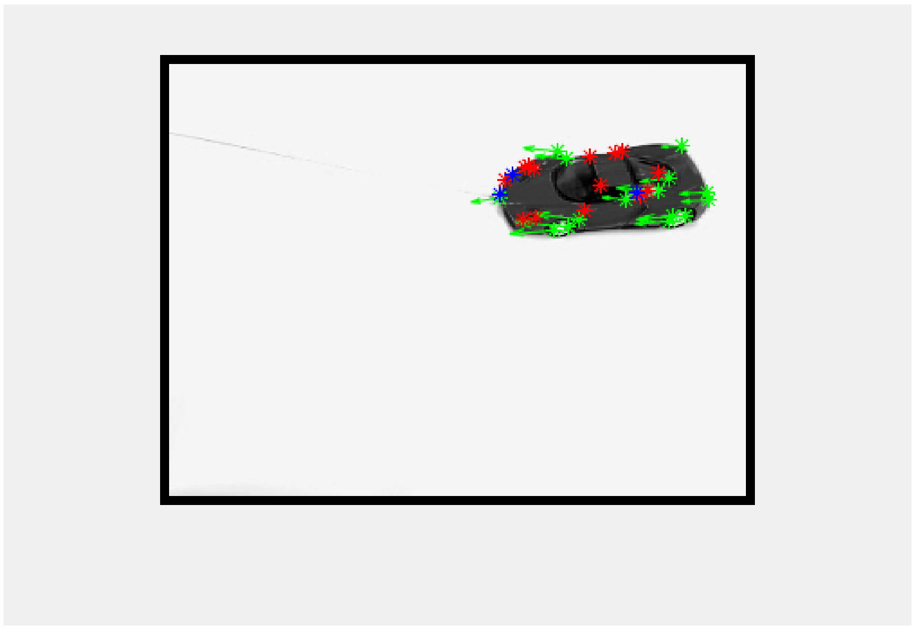

Figure 3: Output video 1, frame # 66, window size: 5x5 pixels

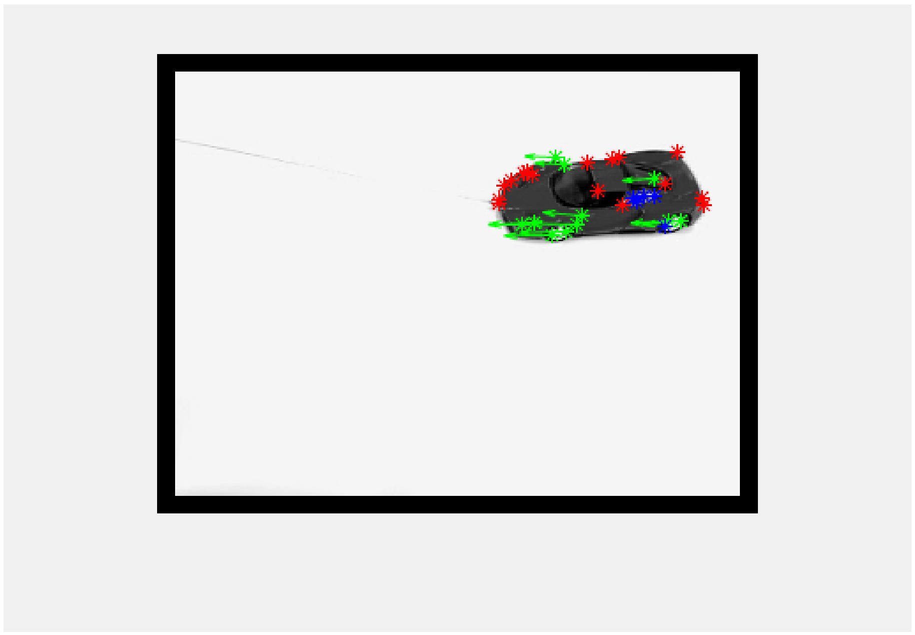

Figure 4: Output video 1, frame # 66, window size: 10x10 pixels

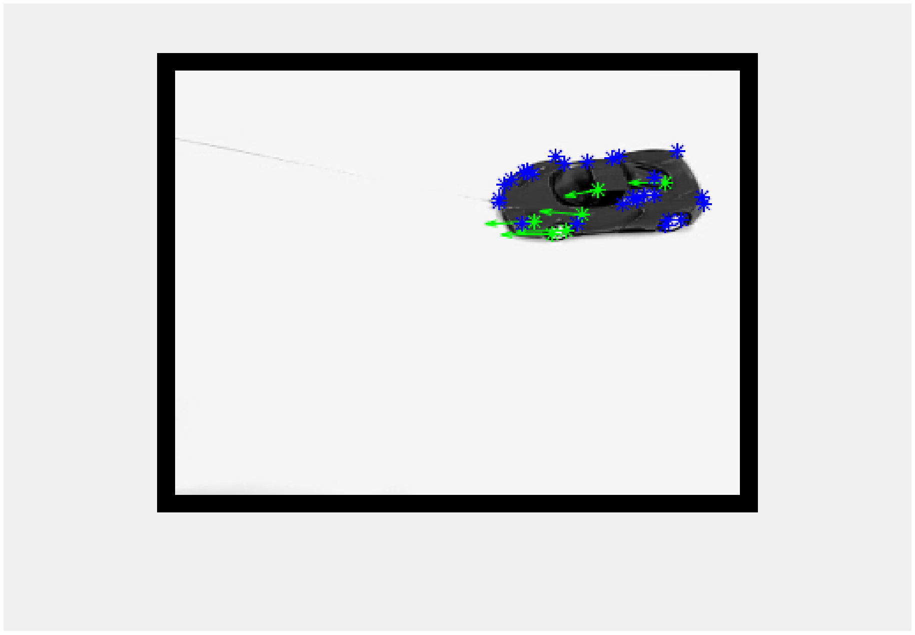

Figure 5: Output video 1, frame # 66, window size: 10x10 pixels, different thresholds

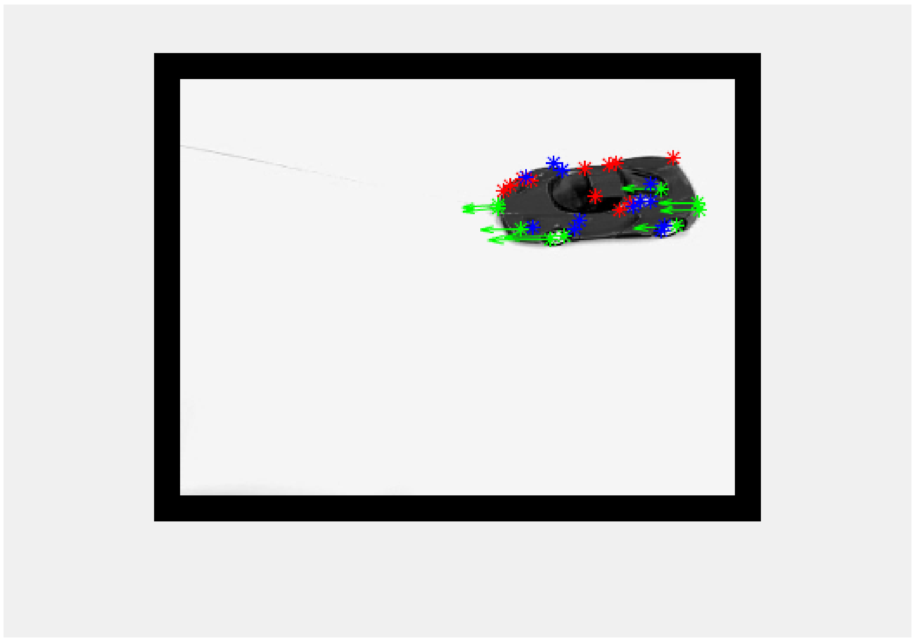

Figure 6: Output video 1, frame # 66, window size: 15x15 pixels

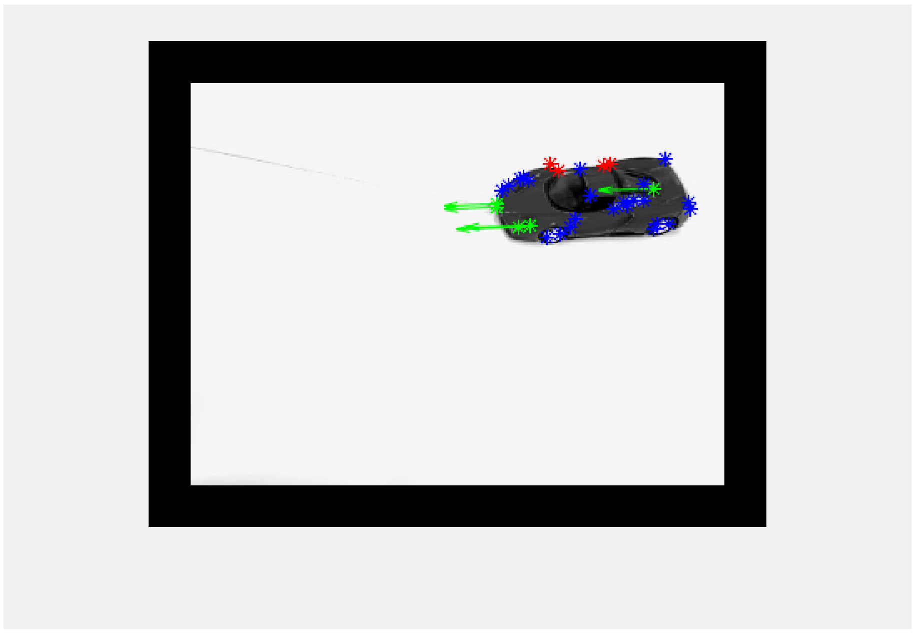
Figure 7: Output video 1, frame # 66, window size: 25x25 pixels

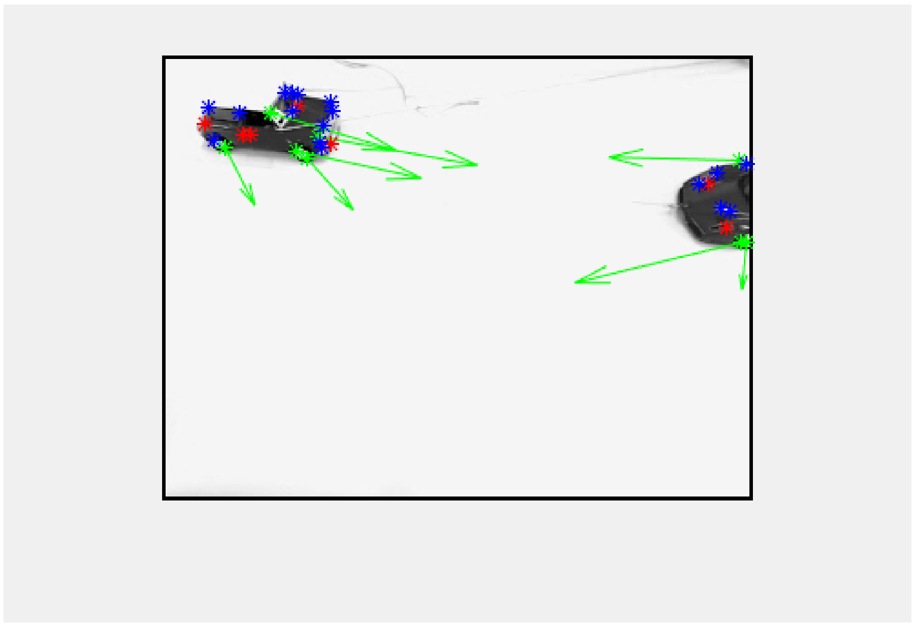
Figure 8: Output video 2

3 Matlab code

```
close all;
clear all;
addpath('lib');
global MAX_FRAMES
global wx
global wy
global threshold1
global threshold2
MAX_FRAMES  = 400;

[movRGB, f_rate] = read_video('../video/car1.avi');
%save('movRGB.mat','movRGB', 'f_rate');
%load('movRGB.mat');

% DEFINING SOME OBJECT WE ARE GOING TO NEED
wx = 25;     % WINDOW SIZE - X
wy = 25;     % WINDOW SIZE - Y
threshold1 = 0.8;    % THRESHOLD ON THE RATIO BETWEEN EIGENVALUES
                     % ( INCREASE TO ADD CORNERS )
threshold2 = 250;    % THRESHOLD ON THE MINIMUM VALUE OF EIGENVALUES
                     % ( INCREASE TO ADD FLAT CORNERS )
f = figure(1);       % FIGURE WITH OUTPUT
[h, w, ~, t] = size(movRGB);     % SIZE OF EACH FRAME
mov = zeros(h+2*wy,w+2*wx,t);    % VAR. FOR GREYSCALE VIDEO
motion = zeros(size(mov,1)+95,size(mov,2)+174,3,t); % VAR. FOR OUTPUT VIDEO
out = cell(t-1,1);               % CELL AR. FOR OUTPUT VALUES
% INITIALIZING MOV WITH THE FIRST FRAME, ZERO PADDED
mov(:,:,1) = zeroPadding(rgb2gray(movRGB(:,:,:,1)/255));
for k=2:t
%for k=2:2
    clearvars outFlow outFlat outEdge;
    % CONVERT RGB FRAME TO GREYSCALE, WITH ZERO PADDING OF THE SIZE OF THE
    % WINDOW
    currInst = zeroPadding(rgb2gray(movRGB(:,:,:,k)/255));
    % SAVE CURRENT GREYSCALE FRAME TO MOV
    mov(:,:,k) = currInst;
    % GET PREVIOUS GREYSCALE FRAME
    prevInst = mov(:,:,k-1);
    % GET CORNERS FROM CURRENT FRAME
    C = corner(currInst,'Harris');
    % DROP CORNERS DUE TO ZERO PADDING
    C = C(5:length(C),:);
    % COMPUTE THE SPATIAL GRADIENT OF THE CURRENT FRAME
    [Ix,Iy] = imgradientxy(currInst);
    % COMPUTE THE TIME GRADIENT BETWEEN THE CURRENT AND THE PREVIOUS FRAMES
    It = (currInst-prevInst)/2;
    j1 = 1;
    j2 = 1;
    j3 = 1;
    % FOR EACH CORNER DETECTED
    for i = 1:size(C,1)
        Cx = C(i,1);
```

```
            Cy = C(i,2);
            % IF IT IS INSIDE THE USEFUL IMAGE
            if ((Cx>wx)&&(Cx<w-wx)&&(Cy>wy)&&(Cy<h-wy))
                % GET THE WINDOWS OF Ix, Iy, It
                IxWin = Ix(Cy-wy:Cy+wy,Cx-wx:Cx+wx);
                IyWin = Iy(Cy-wy:Cy+wy,Cx-wx:Cx+wx);
                ItWin = It(Cy-wy:Cy+wy,Cx-wx:Cx+wx);
                % COMPUTE THE TWO COLUMN OF MATRIX A
                A1stCol = reshape(transpose(IxWin),(2*wx+1)*(2*wy+1),1);
                A2ndCol = reshape(transpose(IyWin),(2*wx+1)*(2*wy+1),1);
                A = [A1stCol,A2ndCol];
                % COMPUTE VECTOR b
                b = reshape(transpose(ItWin),(2*wx+1)*(2*wy+1),1);
                % COMPUTE A^T A AND A^T b
                ATA = transpose(A)*A;
                ATb = transpose(A)*b;
                % GET AND SORT EIGENVALUES OF A^T A
                lambda = sort(eig(ATA),'descend');
                % IF EIG. ARE BIG ENOUGH AND THE RATIO IS SMALL IT'S A CORNER
                if abs(lambda(1)/lambda(2)-1)<threshold1...
                        && lambda(1)>threshold2...
                        && lambda(2)>threshold2
                    % SAVE [CORNER COORD. X, Y, MOTION ESTIMATION X, Y]
                    outFlow(j1,:) = [Cx,Cy,transpose(ATA\(-ATb))];
                    j1 = j1+1;
                % IF EIG. ARE BIG ENOUGH AND THE RATIO ISN'T SMALL IT'S AN EDGE
                else if (lambda(1)>threshold2)&&(lambda(2)>threshold2)
                        outEdge(j3,:) = [Cx,Cy];
                        j3 = j3 + 1;
                % IF EIG. ARE NOT BIG ENOUGH IT'S A FLAT REGION
                    else
                        outFlat(j2,:) = [Cx,Cy];
                        j2 = j2 + 1;
                    end
                end
            end
        end
        imshow(currInst);
        hold on
        if exist('outFlow','var') && size(outFlow,1)>0
            plot(outFlow(:,1),outFlow(:,2),'g*');   % plot corners
            const = 100;    % plot vectors (multiplied by a constant term =100)
            quiver(outFlow(:,1),outFlow(:,2),...
                const*outFlow(:,3),const*outFlow(:,4),'color','g');
        end
        if exist('outFlat','var') && size(outFlat,1)>0
            plot(outFlat(:,1),outFlat(:,2),'r*'); % plot flat region corners
        end
        if exist('outEdge','var') && size(outEdge,1)>0
            plot(outEdge(:,1),outEdge(:,2),'b*'); % plot edge region corners
        end
        motion(:,:,:,k)=frame2im(getframe(f)); % save the computed motion
        hold off
        % save the output to a cell array
        if j1>1, out{k-1,1} = outFlow; else out{k,1} = []; end
        if j2>1, out{k-1,2} = outFlat; else out{k,2} = []; end
```

```
    if j3>1, out{k-1,3} = outEdge; else out{k,3} = []; end
end
close all
% play the results
implay(motion/255, f_rate);
```

which uses function zeroPadding:

```
function out = zeroPadding(in)
global wx
global wy
[h, w] = size(in);
out = [zeros(h+2*wy,wx),...
    [zeros(wy,w);in;zeros(wy,w)],...
    zeros(h+2*wy,wx)];
```