YINSHI ZHANG
University of California, San Diego

December 18, 2014

ASSIGNMENT 4 — LaTeX

Problem 1. Optical Flow

(a) **Dense Optical Flow**

This problem is to implement the single-scale Lucas-Kanade optical flow algorithm. 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. Implementaion function function [u, v, hitMap] = opticalFlow(I1,I2,windowSize, tau) is listed in appendix. Here, to prevent uninvertibal $A^T A$, we need to check if it has rank 2. This is because when the 22 matrix is singular, it present edge pixels where all gradient vectors point in the same direction, which cannot provide accurate information about motion since we cannot tell orientation along the edge. In addition, the gradients have very small magnitude in a low texture region, so the optical flow in these regions is negligible. Therefore, if the smallest eigenvalues of the left matrix of the equation is smaller than a threshold value τ we do not compute the optical flow. Since size of test images are not the same, we tune parameter τ and window size to test optimal setting. Sythentic [windowSize=5,10,15;τ=0.1], sphere [windowSize=10,20,30,τ=0.05], corridor [windowSize=15,30,100; τ=0.02] Figure 1,2,3 shows result for sphere, sythetic image and corridor respectively. Constant gray image presents hitmap with constant value 1, which means no pixel is discarded.

(b) **Corner Detection**

In this part, we implemented corner detection which serves as feature points for sparse optical flow. The results are shown on the first column 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.

(c) **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 second column of Figs 4 5 6, respectively. We tried different windowsizes and found that when windowsize = 100 generates best result.

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 6) 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.

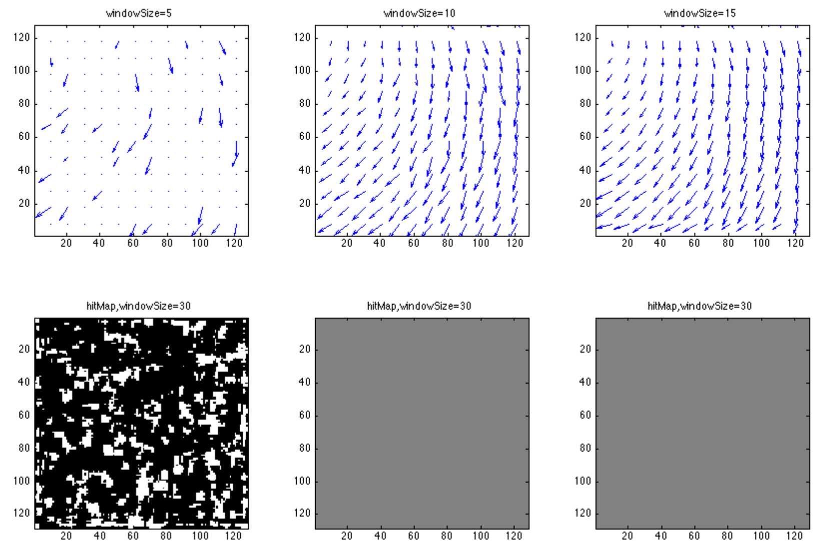


Figure 1: Result for the dense optical flow problem on the sythentic image (gray image shows hitmap with all pixel value=1)

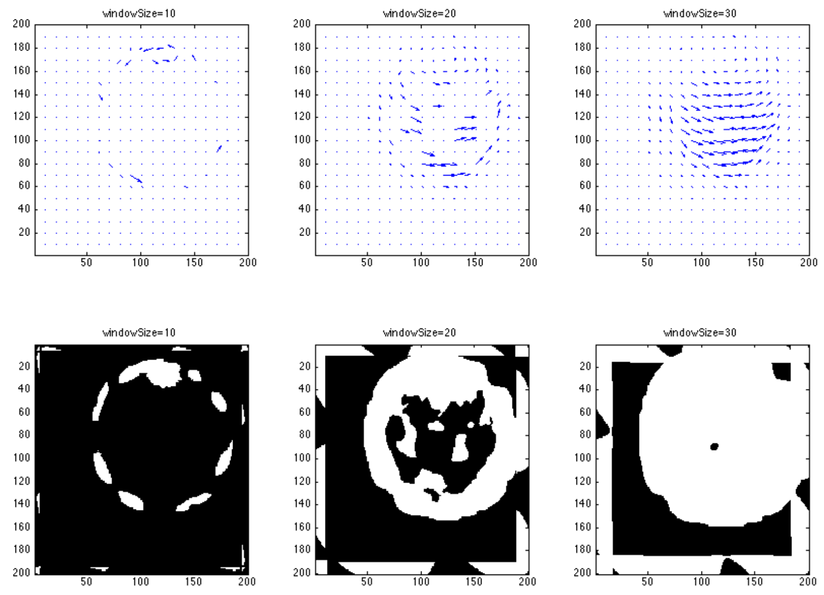


Figure 2: Result for the dense optical flow problem on the sphere image.

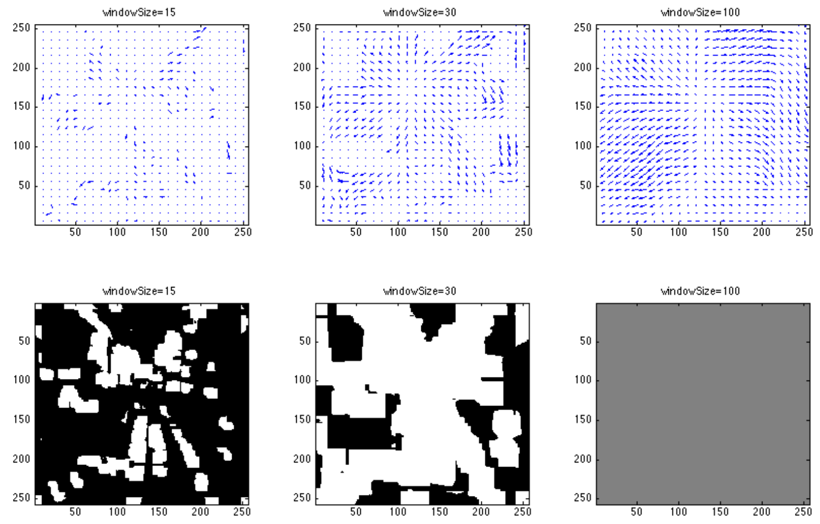


Figure 3: Result for the dense optical flow problem on the corridor image.

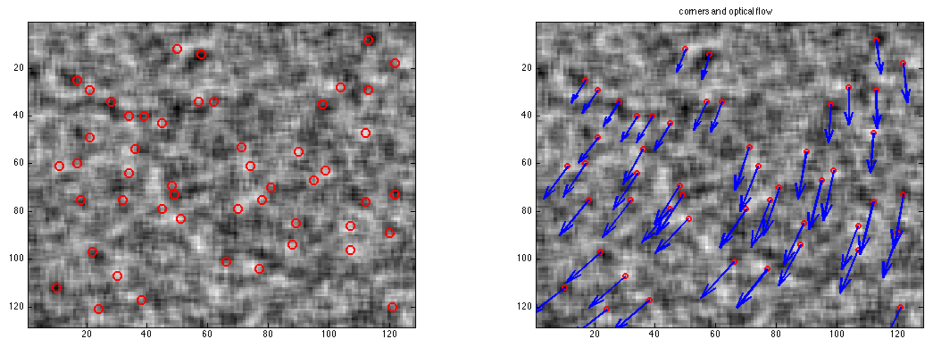


Figure 4: Result for the sparse optical flow problem on the sythentic image.

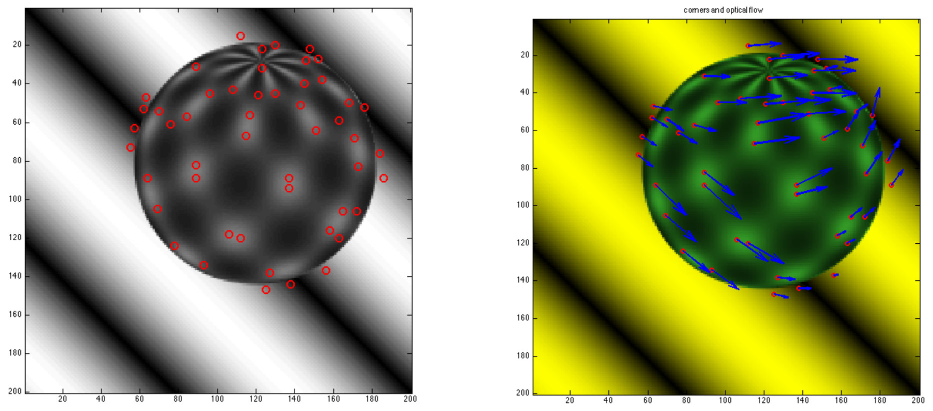


Figure 5: Result for the sparse optical flow problem on the sphere image.

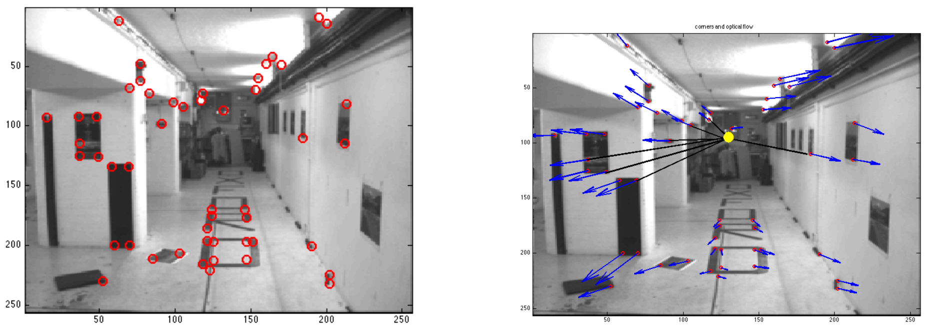


Figure 6: Result for the sparse optical flow problem on the corridor image.

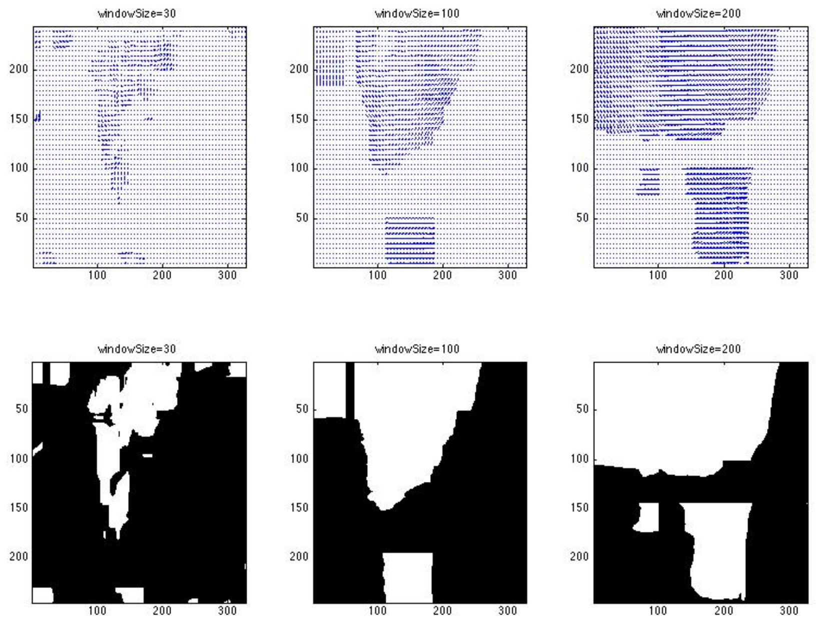


Figure 7: Result for the dense optical flow problem on the image of PC

(d) Own Image

Figure 7 and figure 8 show results for my own image shot in Price Center. The window size is set as [30,100,200] since the image pair I took are not picked from video sequence that the movement is much larger than normal frames.

Problem 2. **Iterative Coarse to Fine Optical Flow**

In this section, we discuss and implement the iterative coarse to fine optical flow algorithm described in the class lecture notes.

The motivation is that, when we apply single-scale Lucas-Kanade optical flow algorithm, we assum a window has little motion so that we can ignored high-order terms in the derivation of taylor expansion. But this sumption fails when if the object we are tracking moves a long distance. In this case, iterative coarse-to-fine method helps a lot.

Iterative coarse to fine optical flow is simply building image pyramids(normally reduce size by 2 times) for each images, and by doing optical flow on each layer of pyramid, to get rid of the small motion constraint. First, we start from the lowest resolution level. We use iterative optical flow to estimate potential motion velocity at this level and then expend it to a higher resolution level as intial velocity for this level to apply lucas-Kanada iteratively. Note that, when we resize estimated motion vector to higher resolution level, we need to multiply two on the estimated vector.

As for iterative process on one level, we firstly estimate potential optical flow on the window corre-

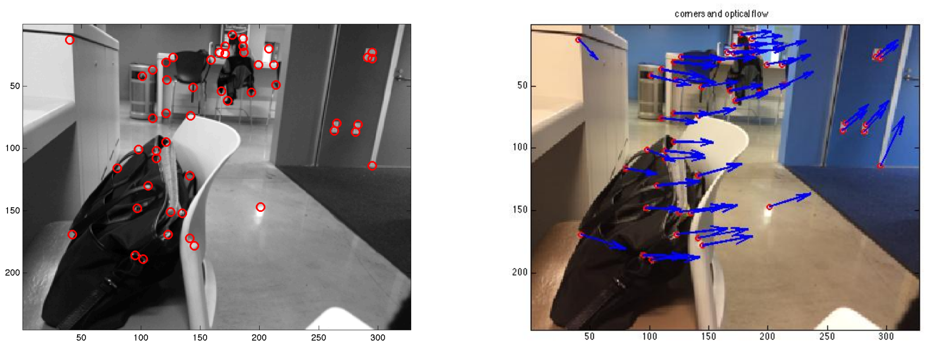


Figure 8: Result for the sparse optical flow problem on the PC image.

sponding to one pixel with simple original Lucas-Kanada, then we reapply the estimated vector to warp image to a new position. We create a warped image as

$$I_0(\overrightarrow{x}, t+\delta) = I(\overrightarrow{x} + \overrightarrow{u}_0\delta t, t+\delta) \tag{1}$$

Next, compute Lucas-Kanada again to estimate residual flow. Then refined optical flow estimation becomes

$$\overrightarrow{u}_1 = \overrightarrow{u}_0 + \delta\hat{u} \tag{2}$$

We repeat this process for several iteration until the residual motion is sufficiently small.

Figure 9. shows result from dense optical flow. with window size of [5,15,30]. Compatively, result from iterative coarse to fine optical flow is figure 9. We can see when window size is 15, we get most accurate flow. Thus, we will use result of windowsize 15 to do segmentation task the last section.

Problem 3. **Bonus: Background Subtraction**

In this problem we intend to extract background from a sequence of video frames, where camera is static. Here we use $|I(x, y, t)I(x, y, t1)| > \tau$ to mask out motion part and then generate a temporal average image as background. Since mean averaging is a linear computing, we can compute iteratively to save memory. Note that, besides remove motion pixels we also need to replace the value with average intensity to maitain global intensity of the image. In experiment, τ is set to 0.001. Figure11 shows final result of highway sequence and truck sequence.

Problem 4. **Bonus: Motion Segmentation**

It is noticed that the magnitudes of the motion vectors at different depths can be quite different. In other words, images of objects that are closer to an observer tend to move faster than images of objects that are farther away. Thus, we can use this cue of distance to segment an image frame of a sequence by threshold on optical flow magnitudes. Here we simply use Euclidean norm $||d|| = \sqrt{u^2 + v^2}$. Figure 12a is result of color map for optical flow magnitudes. Figure 12 shows result of segmentation with cluster number of 2 which segment out the tree and 3 which segment image into 3 parts, tree, flower and sky.

Appendix

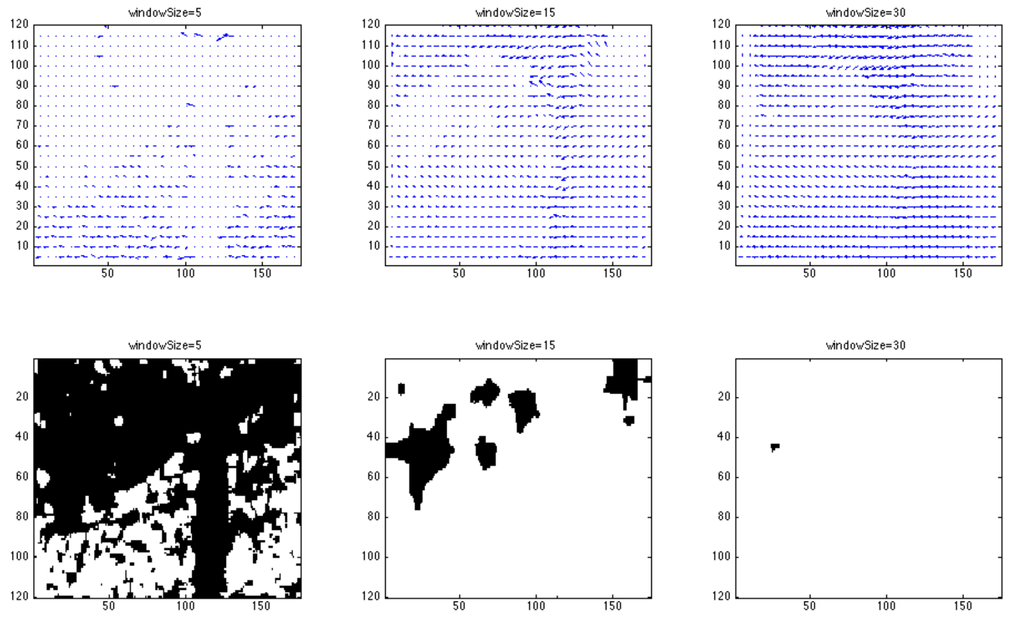


Figure 9: Result of dense optical flow on flower sequence

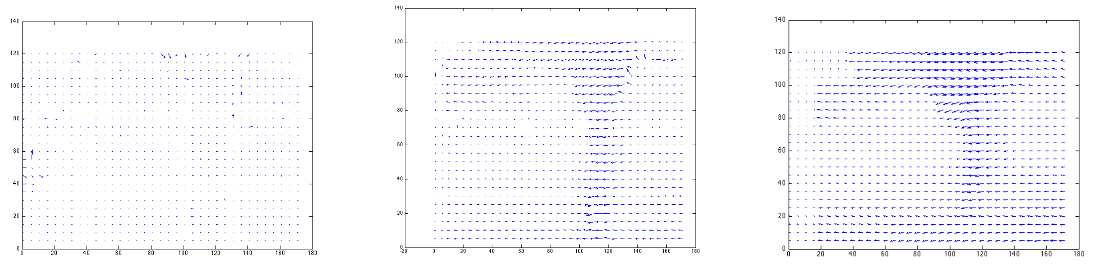
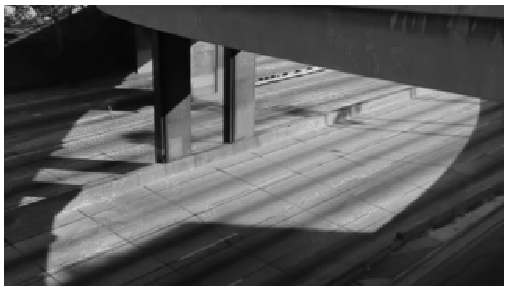
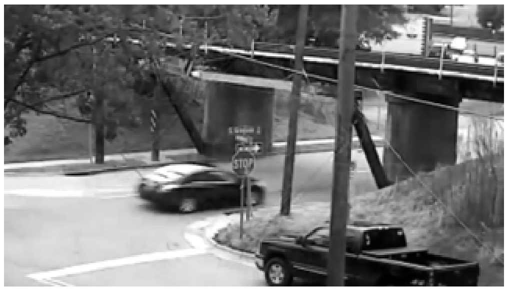


Figure 10: Result of iterative coarse to fine optical flow on flower sequence with windowSize [5,15,30]

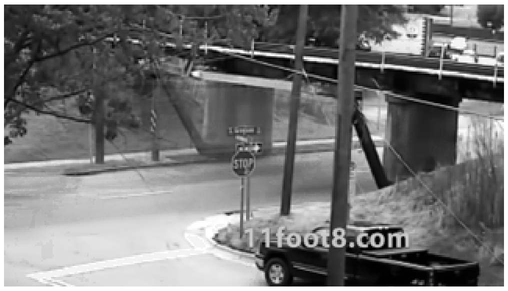
(a) A frame from the highway sequence



(b) Background result for highway sequence



(c) A frame from the truck sequence

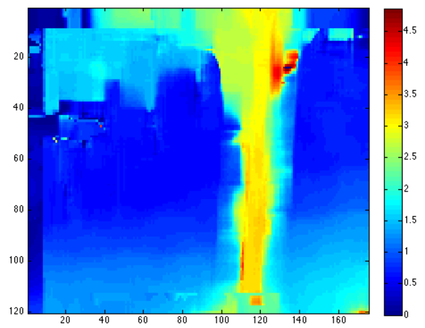


(d) Background result for truck sequence

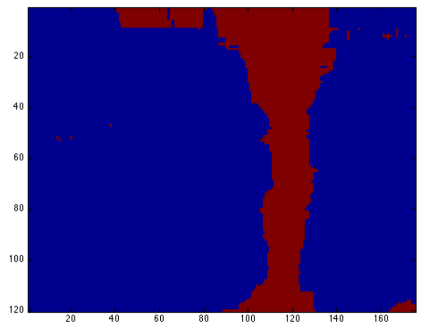
Figure 11: Result of background extraction

Listing 1: Source code for Dense Optical Flow

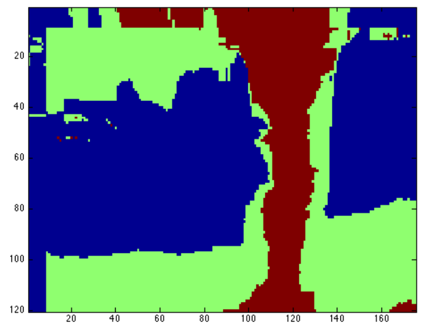
```
function [u, v, hitMap] = opticalFlow(I1,I2,windowSize, tau)
% clear
% close all

% I1 = imread('flower/00029.png');
% I2 = imread('flower/00030.png');
% tau = 0.01;
% windowSize = 15;

if size(I1,3) == 3
        I1 = rgb2gray(I1);
end
I1 = mat2gray(I1);
if size(I2,3) == 3
        I2 = rgb2gray(I2);
end
I2 = mat2gray(I2);

h = fspecial('gaussian', [3,3], 1);
I1 = imfilter(I1,h);
```



(a) color map for optical flow magnitudes



(b) Segment out the tree



(c) Segment into 3 classes

Figure 12: Result of segmentation

```
I2 = imfilter(I2,h);

d = 1/12.*[-1,8,0,-8,1];
Ix = conv2(I1, d,'same');
Iy = conv2(I1,d', 'same');
It = I2-I1;

sz = size(I1);

X2 = conv2(Ix.^2, ones(windowSize),'same');
Y2 = conv2(Iy.^2, ones(windowSize),'same');
XY = conv2(Ix.*Iy, ones(windowSize),'same');
XT = conv2(Ix.*It, ones(windowSize),'same');
YT = conv2(Iy.*It, ones(windowSize),'same');


hitMap = zeros(sz);
u = zeros(sz);
v = zeros(sz);
hsz = floor(windowSize/2);
for i = 1:sz(1)
    for j = 1:sz(2)
        left = j-hsz; right = j+hsz;
        top = i-hsz; bottom = i+hsz;
        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
        ws = (right-left+1)*(bottom-top+1);

        A = [X2(i,j) XY(i,j); XY(i,j) Y2(i,j)]/ws;

        B = (-1).*[XT(i,j); YT(i,j)]/ws;
        r = rank(A);
        lambda = eig(A);
        if (min(lambda)>tau)
            hitMap(i,j) = 1;
            U = A\B;
            u(i,j) = U(1);
            v(i,j) = U(2);
        end

    end
end


```

```
end
```

Listing 2: Script code for experiment and sparse optical flow

```
clear,
tau = 0.2;
% I1 = imread('synth/synth_000.png');
% I2 = imread('synth/synth_001.png');
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('flower/00029.png');
% I2 = imread('flower/00030.png');
windowSize = [15,30,100];

%% optical flow
res=cell(3,2);
for w=1:3
    wsize = windowSize(w);
    [u, v, hitMap] = opticalFlow(I1,I2,wsize, tau);
    saveas(gca, ['map',w,'.png']);

    [x, y] = meshgrid(1:10:size(I1,1), size(I1,1):-10:1);

    qu = u(1:10:size(I1,1), 1:10:size(I1,2));
    qv = v(1:10:size(I1,1), 1:10:size(I1,2));

    subplot(2,3,w), quiver(x,y, qu, -qv,'linewidth', 1),axis([1,size(I1,1)
        ,1,size(I1,2)]), title(sprintf('windowSize=%d',wsize));
    subplot(2,3,w+3), imagesc(hitMap),colormap(gray), title(sprintf('
        windowSize=%d',wsize));

end

%% sparse corners
corners = CornerDetect(I1, 50, 1, 7);
saveas(gca, ['corners.png']);
[u, v, hitMap] = opticalFlow(I1,I2,100, tau);
for i=1:50
    c(i,1)=u(corners(i,1),corners(i,2));
    c(i,2)=v(corners(i,1),corners(i,2));
end
figure;
imagesc(I1);colormap gray
hold on;
```

```
plot(corners(:,2), corners(:,1), 'ro', 'MarkerSize', 7, 'linewidth',2),
    title('corners and optical flow');

hold on;
quiver(corners(:,2),corners(:,1),c(:,1),c(:,2),1,'linewidth',2);
```

Listing 3: Source code for function of single level iterative optical flow

```
function [FU,FV] = iterOpticalflow(I1,I2,initu,initv,windowSize,tau,MAXI)

if size(I1,3) == 3
        I1 = rgb2gray(I1);
end
I1 = mat2gray(I1);
if size(I2,3) == 3
        I2 = rgb2gray(I2);
end
I2 = mat2gray(I2);

% h = fspecial('gaussian', [3,3], 1);
% I1 = imfilter(I1,h);
% I2 = imfilter(I2,h);

d = 1/12.*[-1,8,0,-8,1];
sz = size(I1);
Ix = conv2(I1, d,'same');
Iy = conv2(I1,d', 'same');

X2 = conv2(Ix.^2, ones(windowSize),'same');
Y2 = conv2(Iy.^2, ones(windowSize),'same');
XY = conv2(Ix.*Iy, ones(windowSize),'same');


%% iterative
FU = zeros(sz);
FV = zeros(sz);
hsz = floor(windowSize/2);
for i = 1:sz(1)
    for j =1:sz(2)
        left = j-hsz; right = j+hsz;
        top = i-hsz; bottom = i+hsz;
        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 = [X2(i,j) XY(i,j); XY(i,j) Y2(i,j)];
%             lambda = eig(A);
            r = rank(A);
            if(r~=2)
                Ainv = zeros(2);
            else
                Ainv = inv(A);
            end
            u=initu(i,j);
            v=initv(i,j);
            for iter = 1:MAXI
                [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 = -1.*[sum(ixt(:)); sum(iyt(:))];
                U = Ainv*B;
                U(isnan(U)) = 0 ;
                u = u+U(1); v = v+U(2);
                if(abs(U(1))<tau && abs(U(2))<tau) break;end
            end
            FU(i,j)=u; FV(i,j)=v;
        end

end

[x, y] = meshgrid(1:5:size(I1,2), size(I1,1):-5:1);
qu = FU(1:5:size(I1,1), 1:5:size(I1,2));
qv = FV(1:5:size(I1,1), 1:5:size(I1,2));
figure,title('dense')
quiver(x,y, qu, -qv,'linewidth', 1);
end
```

Listing 4: Script code for iterative coarse to fine optical flow

```
clear
close all

```

```
I1 = imread('flower/00029.png');
I2 = imread('flower/00030.png');
%% initial & build pyramid
MAXI=10;
tau=0.01;
windowSize = 15;
% d = 1/12.*[-1,8,0,-8,1];

if size(I1,3) == 3
        I1 = rgb2gray(I1);
end
I1 = mat2gray(I1);
if size(I2,3) == 3
        I2 = rgb2gray(I2);
end
I2 = mat2gray(I2);


i1 = cell(3,1);
i1{3} = I1;
i1{2} = impyramid(I1, 'reduce');
i1{1} = impyramid(i1{2}, 'reduce');


i2 = cell(3,1);
i2{3} = I2;
i2{2} = impyramid(I2, 'reduce');
i2{1} = impyramid(i2{2}, 'reduce');


%% iteration optical flow
% initial
initu=zeros(size(i1{1})); initv=zeros(size(i1{1}));
[u0, v0] = iterOpticalflow(i1{1},i2{1},initu, initv, windowSize,tau, MAXI)
    ;

% iterative
tempu = u0; tempv = v0;
for l = 2:3
    upu = imresize(tempu, size(i1{l}));
    upv = imresize(tempv, size(i1{l}));
    upu = 2.*upu;
    upv = 2.*upv;
    [tempu, tempv]=iterOpticalflow(i1{l},i2{l},upu, upv, windowSize,tau,
        MAXI);
end
```

```


resu = tempu;
resv = tempv;

[x, y] = meshgrid(1:5:size(I1,2), size(I1,1):-5:1);
qu = resu(1:5:size(I1,1), 1:5:size(I1,2));
qv = resv(1:5:size(I1,1), 1:5:size(I1,2));
figure,title('coarse to fine')
quiver(x,y, qu, -qv,'linewidth', 1);

%% segmentation
map = zeros(size(resu));
for i=1:size(resu,1)
    for j=1:size(resu,2)
        map(i,j) =sqrt((resu(i,j)).^2 + (resv(i,j)).^2);

    end
end

map(map>5)=0
figure, imagesc(map),title('pyramid');
```

Listing 5: Source code of background extraction

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

% allFiles = dir(fullfile( 'truck','*.png' ));
% allFiles = dir(fullfile( 'truck','*.png' ));
% tau = 0.0001;
frame = cell(size(framesequence));

num = size(allFiles,1);
for n=1:num
    frame{n} = im2double(imread(fullfile('truck',allFiles(n,1).name)));

end
% bg = cell(num);
temp = zeros(size(frame{1}));
for i = 2:num
    mask = (frame{i}-frame{i-1})<tau;
        frame{i}(mask) = 0;
    red = zeros(size(temp));
    red(mask)=temp(mask);
    temp = (temp.*(i-2)+frame{i}+red)./(i-1);
end
```

```
imshow(temp);
imwrite(temp, 'back_t.png');

end
```

Submitted by YINSHI ZHANG on December 18, 2014.