Homework 4
Project Report
Optical Flow

FNU KARAN
NetID: kx361

April 28, 2017

Date Performed: April 23, 2017
Instructor: Professor Gerig

1 Objective

The objective is to come up with the optical flow between two consecutive images of a video sequence.

2 Data Capture

The images from the video sample have been provided with the problem statement. Some of the image samples are shown in Figure 1.

Figure 1: Input Images from the video sequence

3 Experimental Procedure

3.1 Optical Flow Theory

The Optical flow method relies on two assumptions that will allow us to derive its equation :
Assumption 1: The motion between frames is small
Assumption 2: Conservation of pixel brightness over time

The second assumption leads to formulate the brightness consistency constraint:

$$\frac{df}{dt} = 0 \implies I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t) \tag{1}$$

The previous equations means that we assume that in this framework, only the object is moving and its brightness properties remain constant over time so that the shading and light parameters are supposed to be the same.
Let's now considerate the change of position of pixel p introduced before, we can then write the following equations:

$$u = \frac{\partial x}{\partial t}, \quad v = \frac{\partial y}{\partial t} \tag{2}$$

So now let's differentiate the f function with Taylor expansion and use the brightness consistency constraint:

$$\frac{dI}{dt} = 0 \implies \frac{\partial I}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial I}{\partial y}\frac{\partial y}{\partial t} + \frac{\partial I}{\partial t} = 0 \quad \implies \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0 \tag{3}$$

The two terms before u and v respectively are the image gradients in the x respectively y direction of the image. So if we introduce the ∇ operator of first order partial derivative :

$$\nabla I = \begin{bmatrix} \frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} \end{bmatrix}^T = \begin{bmatrix} I_x \\ I_y \end{bmatrix} \tag{4}$$

and finally if we define the displacement $\vec{v}$ such as:

$$\vec{v} = \begin{bmatrix} \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} \end{bmatrix}^T = \begin{bmatrix} u \\ v \end{bmatrix} \tag{5}$$

We can formulate the compact form equation of Optical Flow:

$$\nabla \vec{v} + I_t = 0 \tag{6}$$

The goal of this method is to determine the unknown $\vec{v} = (u, v)$, because the time and spatial gradients of the image can be computed and here we try to estimate motion.

In this method, we rely on estimation of the image gradients in both time and space variables. There is one point to discuss regarding the computation of spatial gradients. Indeed, the spatial gradient being a derivative, if we have a very sharp and straight edge, our gradient computation can fail. To avoid this issue, we are going to introduce a pre processing step that consists in a Gaussian smoothing of the image to smooth edges.

An other aspect will be discussed later and deals with the computational technique to estimate the spatial gradients of the image. Indeed, the time gradient is easily defined by:

$$I_t = \frac{\partial I}{\partial t} = I^*(x, t + \delta t) - I^*(x, t) \tag{7}$$

where I^* is the smoothed image.
But there are several ways to compute image gradients here are two examples to compute them :

$$I_x = \frac{\partial I}{\partial x} = I^*(x + \delta x, y, t) - I^*(x, y, t) \quad or \quad I_x = \frac{\partial I}{\partial x} = I^*(x + \delta x, y, t) - I^*(x - \delta x, y, t) \tag{8}$$

3.2 Normal Flow

In this method we only deal with pixel information and do not take in account the neighborhood information. As we mentioned in the theoretical part, this could lead to an aperture problem where the overall motion of the object is not taken in account. We only compute for each pixel an estimation linked to the three gradients available. Here is the definition of the normal flow we can compute thanks to a single pixel information:

$$\vec{v_n} = \frac{-I_t}{||\nabla I||}\frac{\nabla I}{||\nabla I||} \quad with \quad ||\nabla I|| = \sqrt{I_x^2 + I_y^2} \tag{9}$$

3.3 Flow calculated over pixel neighborhood

We choose 2×2 pixel neighborhoods for a local estimate of velocities using the following solution strategy:

$$\nabla E.v + \frac{\partial E}{\partial t} = 0 \tag{10}$$

Now, we can take 4 measurements from 2×2 pixels, and we can then solve for v:

$$Av + b = 0$$

$$A^T Av = -A^T b$$

$$v = -(A^T A)^{-1} A^T A$$

The columns of A are the x and y components of the gradient ∇E and b is a column vector of the t gradient component of E, E_t.

3.4 Horn and Schunck Algorithm

This method relies on the use of partial differential equations and the use of optimization techniques to estimate motion parameters. In this method we want to optimize the following energy minimization problem. Let E be our energy function defined by a sum of an error constraint function and a smoothness constraint on integrability:

$$\mathbf{E} = E_c + \lambda E_s \quad with \quad \lambda \in \mathbb{R} \tag{11}$$

where E_c and E_s are:

$$E_c = \int\int_{\Omega} (I_x u + I_y v + I_t)^2 \, dx \, dy \tag{12}$$

$$E_s = \int\int_{\Omega} ((U_x^2 + U_y^2) + (V_x^2 + V_y^2)) \, dx \, dy \tag{13}$$

where I_x, I_y and I_t are the gradient of the image previously defined and $U_x = \frac{\partial U}{\partial x}$ and $U_y = \frac{\partial U}{\partial y}$ and respectively for V.
We are looking for an estimation of the parameters u and v which minimize E so we apply the classical gradient descent method. So we need to differentiate E with respect to those two parameters as follow:

$$\begin{aligned} \frac{\partial E}{\partial u} &= 2I_x(I_x u + I_y v + I_t) + \lambda(2U_{xx} + 2U_{yy}) \\ \frac{\partial E}{\partial v} &= 2I_y(I_x u + I_y v + I_t) + \lambda(2V_{xx} + 2V_{yy}) \end{aligned} \tag{14}$$

So to implement and solve this problem here is the estimation of u and v parameters:

$$\begin{aligned} u^{n+1} &= u^n - I_x \frac{I_x \bar{u}^n + I_y \bar{v}^n + I_t}{1 + \lambda(I_x^2 + I_y^2)} \\ v^{n+1} &= v^n - I_x \frac{I_x \bar{u}^n + I_y \bar{v}^n + I_t}{1 + \lambda(I_x^2 + I_y^2)} \end{aligned} \tag{15}$$

4 Results

4.1 Normal Vectors

Here are the results obtained for synthetic images after coding the above explained algorithm in MATLAB. We get the derivatives in x and y as shown in Figure 2, corresponding temporal derivative is shown in Figure 3 and the normal vectors are shown in Figure 4.

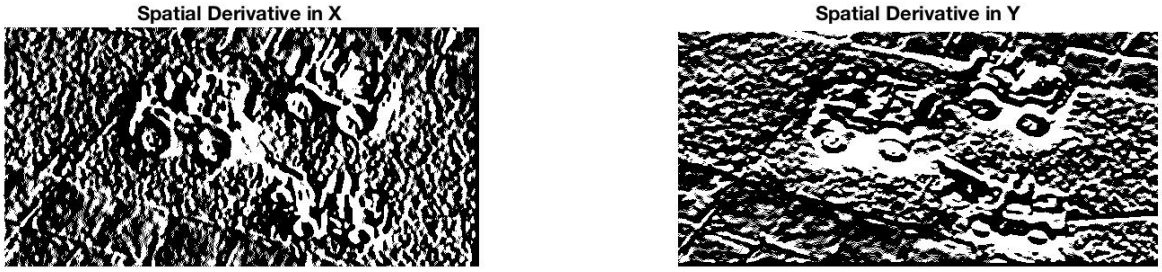


Figure 2: Spatial derivatives in x and y directions

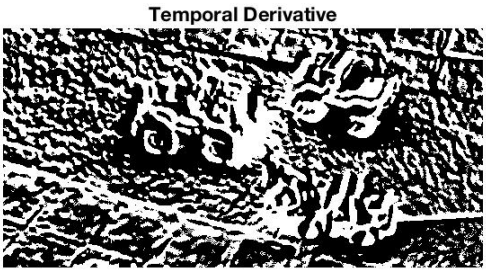


Figure 3: Temporal Derivative

(a) Indvidual u and v vectors

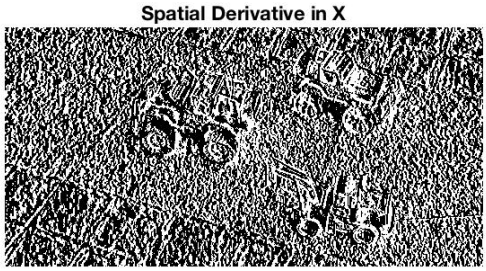
(b) Combined u and v vectors

Figure 4: Normal Vectors

4.2 Flow calculated over pixel neighborhood

4.2.1 Calculation on Raw Images

Here are the results obtained for synthetic images after coding the flow calculation algorithm over pixel neighborhood in MATLAB. We get the derivatives in x and y as shown in Figure 5, corresponding temporal derivative is shown in Figure 6 and the velocity vectors are shown in Figure 7.



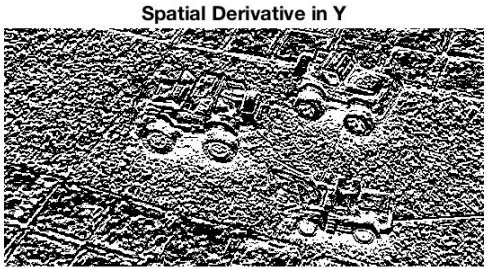


Figure 5: Spatial derivatives in x and y directions

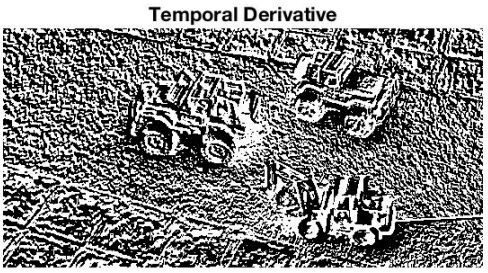


Figure 6: Temporal Derivative

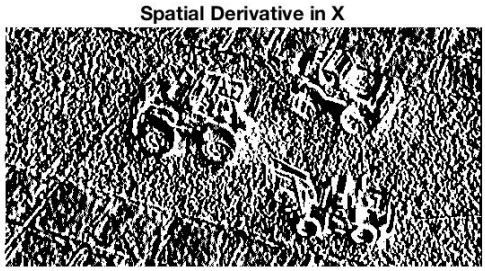
(a) Indvidual u and v vectors

(b) Combined u and v vectors

Figure 7: Normal Vectors

4.2.2 Calculation of Smoothened images over Gaussian filter with $\sigma = 1$

Here are the results obtained for synthetic images after coding the flow calculation algorithm over pixel neighborhood in MATLAB after smoothening the images with σ =1. We get the derivatives in x and y as shown in Figure 8, corresponding temporal derivative is shown in Figure 9 and the flow vectors are shown in Figure 10.



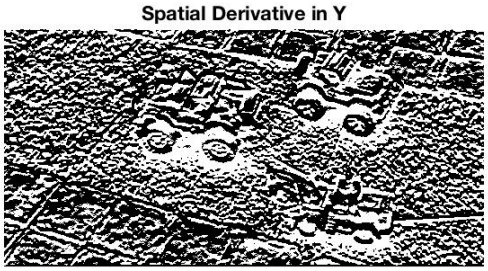


Figure 8: Spatial derivatives in x and y directions

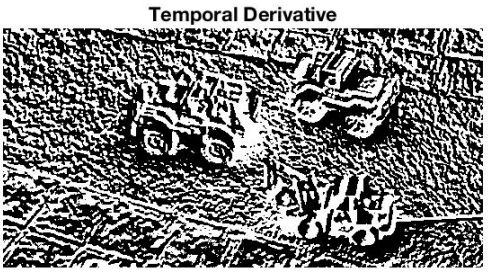


Figure 9: Temporal Derivative

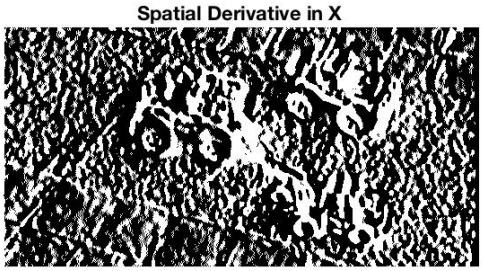
(a) Indvidual u and v vectors

(b) Combined u and v vectors

Figure 10: Velocity Vectors with $\sigma = 1$

4.2.3 Calculation of Smoothened images over Gaussian filter with $\sigma = 2$

Here are the results obtained for synthetic images after coding the flow calculation algorithm over pixel neighborhood in MATLAB after smoothening the images with σ =2. We get the derivatives in x and y as shown in Figure 11, corresponding temporal derivative is shown in Figure 12 and the flow vectors are shown in Figure 13.



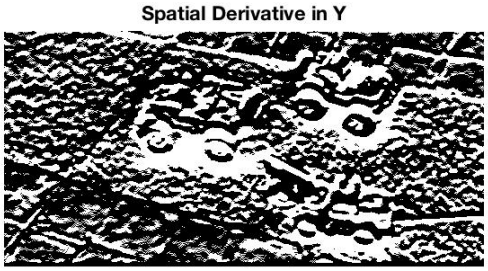


Figure 11: Spatial derivatives in x and y directions

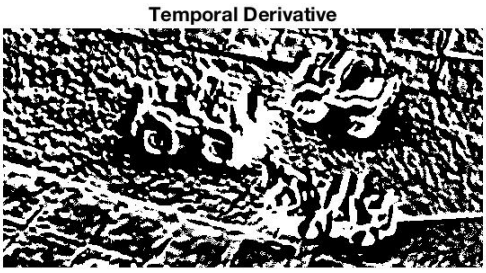


Figure 12: Temporal Derivative

(a) Indvidual u and v vectors

(b) Combined u and v vectors

Figure 13: Velocity vectors with $\sigma = 2$

4.3 Horn and Schunck Algorithm

We obtain the following flow vectors after running the Horn and Schunck Algorithm, as shown in figure 14.

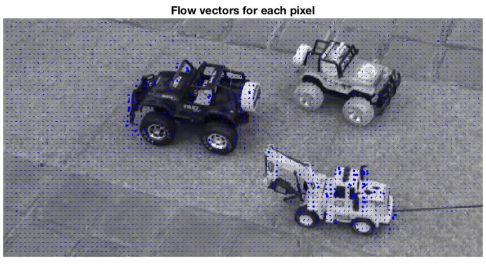


Figure 14: Flow vectors from Horn and Schunck Algorithm

4.4 Our own input sequence

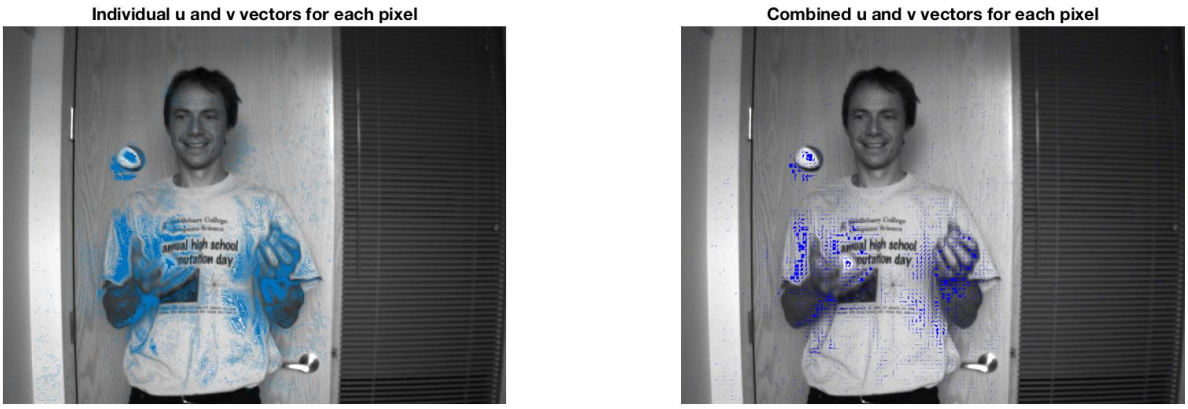
I obtained a few of the input data from *http://vision.middlebury.edu/flow/data/* and I tried to run my algorithm on that dataset. Here are the input images,

(a) Indvidual u and v vectors (b) Combined u and v vectors

Figure 15: Input images

The results are shown below.

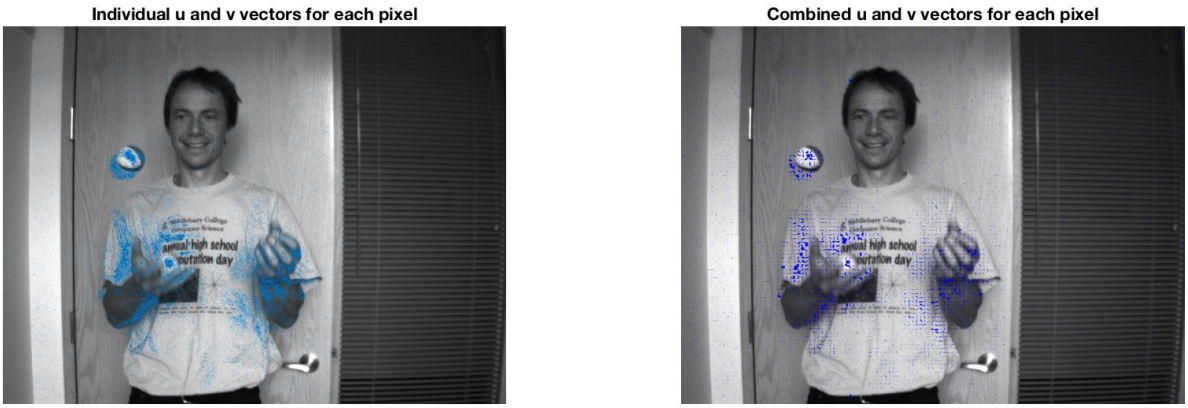
4.4.1 Normal Vectors



(a) Indvidual u and v vectors (b) Combined u and v vectors

Figure 16: Normal Vectors

4.4.2 Flow calculated over pixel neighborhood



(a) Indvidual u and v vectors

(b) Combined u and v vectors

Figure 17: Flow Vectors after smoothening the images with $\sigma = 2$

4.4.3 Horn and Schunck Algorithm

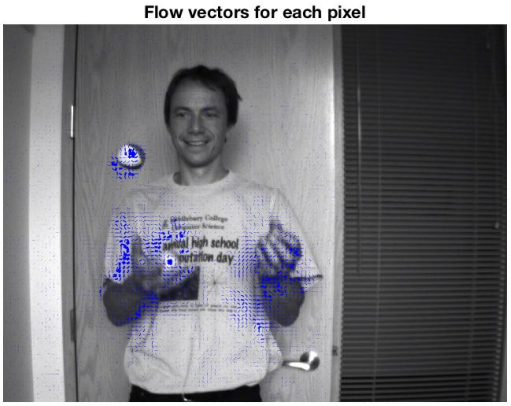


Figure 18: Flow vectors from Horn and Schunck Algorithm

5 Discussion

5.1 Synthetic Images

For the results of synthetic images, first two images were used. We overlay the vector field on the original image to show that it correctly estimates the displacement of the objects of the scene even if we can already see it on the raw field. And as we can see it also seems here that our Horn and Schunck implementation is also succeeding in reconstructing the optical flow for the motion of the three cars. When we compare it to the flow calculated using neighborhood, we see that the latter is much better, since it assumes the 2x2 pixels to have the same flow, thus representing a strong field.

Some improvement can then be discussed. Indeed, in this implementation we only considered a standard 4 connected neighbourhood, but we could also have a weighted 8 connected neighborhood. This introduces some slight differences.

One other aspect we can quickly discuss is the inter frames motion. Indeed, we explained that motion between frames as to be small enough to have a nice working solution. The pre processing smoothing also influenced a bit this aspect, because by smoothing more our images and according to the method we use to solve and reconstruct the displacement field, we are still able to reconstruct the displacement field even in case of larger displacement.

5.2 Our Images

The results are pretty obvious. We can see that the guy in the image is juggling, and by looking at the flow field in the respective methods, we can clearly see that the best results are obtained from Horn and Schunck Algorithm, when compared to the other two methods.

6 MATLAB Code

The code is self explanatory and can be understood with the comments written in the code.

6.1 Calculation of Normal Vectors

```
%% Clear the environment
clc;
clear;

%% Read the images
im1 = imread('/Users/kchak/Google Drive/NYU/Spring 2017/Computer Vision/Assignment
    /4/toy-car-images-bw/toy_formatted2.png');
im2 = imread('/Users/kchak/Google Drive/NYU/Spring 2017/Computer Vision/Assignment
    /4/toy-car-images-bw/toy_formatted3.png');

% Uncomment for reading our own images.
% im1=imread('/Users/kchak/Google Drive/NYU/Spring 2017/Computer Vision/Assignment
    /4/beanbags/frame08.png');
% im2=imread('/Users/kchak/Google Drive/NYU/Spring 2017/Computer Vision/Assignment
    /4/beanbags/frame09.png');
%% Smoothen images
i1_double = double(im1);
i2_double = double(im2);

i1_smoothed = double(gaussian_filter(im1, 2.0));
i2_smoothed = double(gaussian_filter(im2, 2.0));

%% Estimate the parameters

[fx, fy, ft] = computeDerivatives(i1_smoothed, i2_smoothed);

figure;
imshow(fx)
title('Spatial Derivative in X');

figure;
imshow(fy);
title('Spatial Derivative in Y');

figure;
imshow(ft);
title('Temporal Derivative');

%% Calculate the Normal Vectors

image_size = size(i1_smoothed);
rows = image_size(:,1);
```

```
cols = image_size(:,2);

spatial_gradient = zeros(rows, cols, 2);
spatial_gradient(:,:,1) = fx;
spatial_gradient(:,:,2) = fy;

normal_vector = zeros(rows, cols, 2);
for i = 1:rows
    for j=1:cols
        temp = [spatial_gradient(i,j,1); spatial_gradient(i,j,2)];
        normal_vector(i,j,:) = - ft(i,j) * ( spatial_gradient(i,j,:) / (norm(temp))
            ^2 );
    end
end

normal_vector(isnan(normal_vector))=0;

%% Plot the vectors
figure;
imshow(im1);
hold on;
quiver(normal_vector(1:1:rows,1:1:cols,1),normal_vector(1:1:rows,1:1:cols,2),5);
title('Individual u and v vectors for each pixel');
plotFlow(normal_vector(:,:,1), normal_vector(:,:,2), im1, 5, 5);
title('Combined u and v vectors for each pixel');
```

6.2 Calculation of Flow vectors using 2x2 grid neighborhood

```
%% Clear the environment
clc;
clear;

%% Read the images
im1 = imread('/Users/kchak/Google Drive/NYU/Spring 2017/Computer Vision/Assignment
    /4/toy-car-images-bw/toy_formatted2.png');
im2 = imread('/Users/kchak/Google Drive/NYU/Spring 2017/Computer Vision/Assignment
    /4/toy-car-images-bw/toy_formatted3.png');

% Uncomment for reading our own images.
% im1=imread('/Users/kchak/Google Drive/NYU/Spring 2017/Computer Vision/Assignment
    /4/beanbags/frame08.png');
% im2=imread('/Users/kchak/Google Drive/NYU/Spring 2017/Computer Vision/Assignment
    /4/beanbags/frame09.png');

%% Smoothen images
i1_double = double(im1);
i2_double = double(im2);

i1_smoothed = double(gaussian_filter(im1, 1.0));
i2_smoothed = double(gaussian_filter(im2, 1.0));

%% Estimate the parameters

[fx, fy, ft] = computeDerivatives(i1_smoothed, i2_smoothed);
% [fx, fy, ft] = computeDerivatives(double(im1), double(im2));
figure;
imshow(fx);
title('Spatial Derivative in X');
figure;
```

```
imshow(fy);
title('Spatial Derivative in Y');
figure;
imshow(ft);
title('Temporal Derivative');

%% Algorithm

image_size = size(i1_smoothed);
rows = image_size(:,1);
cols = image_size(:,2);

spatial_gradient = zeros(rows, cols, 2);
spatial_gradient(:,:,1) = fx;
spatial_gradient(:,:,2) = fy;

velocity_vector = zeros(rows,cols,2);

for i=1:rows-1
    for j= 1:cols-1

        A = [
                fx(i,j),fy(i,j);
                fx(i,j+1),fy(i,j+1);
                fx(i+1,j),fy(i+1,j);
                fx(i+1,j+1),fy(i+1,j+1)
            ];

        b = [
                ft(i,j);
                ft(i,j+1);
                ft(i+1,j);
                ft(i+1,j+1)
            ];

        C = pinv(A'*A);

        velocity_vector(i,j,:) = -C*A'*b;

    end;
end;

velocity_vector(isnan(velocity_vector))=0;

%% Plot the vectors
figure;
imshow(im1);
hold on;
quiver(velocity_vector(1:1:rows,1:1:cols,1),velocity_vector(1:1:rows,1:1:cols,2),5);
title('Individual u and v vectors for each pixel');
plotFlow(velocity_vector(:,:,1), velocity_vector(:,:,2), im1, 5, 5);
title('Combined u and v vectors for each pixel');
```

6.3 Horn and Schunk Algorithm Implementation

```
%% Clear the environment
clc;
clear;

```

```
%% Read the images
im1=imread('/Users/kchak/Google Drive/NYU/Spring 2017/Computer Vision/Assignment/4/toy-car-images-bw/toy_formatted2.png');
im2=imread('/Users/kchak/Google Drive/NYU/Spring 2017/Computer Vision/Assignment/4/toy-car-images-bw/toy_formatted3.png');

% Uncomment for reading our own images.
% im1=imread('/Users/kchak/Google Drive/NYU/Spring 2017/Computer Vision/Assignment/4/beanbags/frame08.png');
% im2=imread('/Users/kchak/Google Drive/NYU/Spring 2017/Computer Vision/Assignment/4/beanbags/frame09.png');

%% Smoothen images

im1=double(im1);
im2=double(im2);

im1_smoothed=double(gaussian_filter(im1,1));
im2_smoothed=double(gaussian_filter(im2,1));

%% Set initial parameters.

% Set initial value for the flow vectors
u = 0;
v = 0;

% Estimate spatiotemporal derivatives
[fx,fy,ft] = computeDerivatives(im1_smoothed,im2_smoothed);

%% HS Algorithm

% Averaging kernel
kernel=[1/12 1/6 1/12;1/6 0 1/6;1/12 1/6 1/12];
alpha=2;
% Iterations
for i=1:200
    % Compute local averages of the flow vectors
    uAvg=conv2(u,kernel,'same');
    vAvg=conv2(v,kernel,'same');
    % Compute flow vectors constrained by its local average and the optical flow constraints
    u= uAvg - ( fx .* ( ( fx .* uAvg ) + ( fy .* vAvg ) + ft ) ) ./ ( alpha^2 + fx .^2 + fy.^2);
    v= vAvg - ( fy .* ( ( fx .* uAvg ) + ( fy .* vAvg ) + ft ) ) ./ ( alpha^2 + fx .^2 + fy.^2);
end

u(isnan(u))=0;
v(isnan(v))=0;

%% Plot the Flow

plotFlow(u, v, im1, 5, 5);
title('Flow vectors for each pixel');
```

6.4 Utility Functions

6.4.1 Computing Derivatives

```
function [fx, fy, ft] = computeDerivatives(i1_smoothed,i2_smoothed)

% FIRST WAY TO CALCULATE DERIVATIVES
image_size = size(i1_smoothed);
rows = image_size(:,1);
cols = image_size(:,2);

ft = i2_smoothed - i1_smoothed;

fx = zeros(rows,cols);

for i = 2:rows
    for j = 1:cols
        fx(i,j) = double(i1_smoothed(i,j) - i1_smoothed(i-1,j));
    end
end

fy = zeros(rows,cols);
for i = 1:rows
    for j = 2:cols
        fy(i,j) = double(i1_smoothed(i,j) - i1_smoothed(i,j-1));
    end
end

% SECOND WAY TO CALCULATE DERIVATIVES
fx = conv2(i1_smoothed,[1 -1]);
fx = fx(:,1:end-1);
fy = conv2(i1_smoothed,[1; -1]);
fy = fy(1:end-1,:);
ft= i2_smoothed-i1_smoothed;
```

6.4.2 Plotting Flow

```
function [fx, fy, ft] = computeDerivatives(i1_smoothed,i2_smoothed)

% FIRST WAY TO CALCULATE DERIVATIVES
image_size = size(i1_smoothed);
rows = image_size(:,1);
cols = image_size(:,2);

ft = i2_smoothed - i1_smoothed;

fx = zeros(rows,cols);

for i = 2:rows
    for j = 1:cols
        fx(i,j) = double(i1_smoothed(i,j) - i1_smoothed(i-1,j));
    end
end

fy = zeros(rows,cols);
for i = 1:rows
    for j = 2:cols
        fy(i,j) = double(i1_smoothed(i,j) - i1_smoothed(i,j-1));
    end
end
```

```
24
25 % SECOND WAY TO CALCULATE DERIVATIVES
26 fx = conv2(i1_smoothed,[1 -1]);
27 fx = fx(:,1:end-1);
28 fy = conv2(i1_smoothed,[1; -1]);
29 fy = fy(1:end-1,:);
30 ft= i2_smoothed-i1_smoothed;
```