Optical Flow Estimation
Project 04 - Report

Usama Munir Sheikh
ECSE 6650: Computer Vision with Dr. Qiang Ji, RPI
Troy, New York
sheiku@rpi.edu

***Abstract/Goals*— The goal of this project was to compute Optical Flow for three image sequences consisting of five consecutive image frames each. Only the implementation of one of the two methods, discussed in class, was required for this project. The first method is the Lucas Kanade Method and the second method is the Image Brightness Change Constancy Method. I have used method II for this Project.**

Keywords- Image Brightness Change Constancy, Optical Flow;

I. Introduction

"Optical flow is a vector field in the image that represents a projection of the image motion field, along the direction of the image intensity gradient."[1]

For this project we have computed optical flow for a set of three image sequences each consisting of five consecutive image frames each. The optical flow estimation is computed for the middle image using all five image frames.

The first set of image sequences is that of a synthetic image of a rotating sphere. The second set is also synthetic which shows a checkerboard like pattern that only has translational motion in the x-direction. The third and final image set shows two people in it. All three image sets have some movement in them and after the optical flow estimation, we will be able to figure out the directional movement vectors of these objects and plot them out.

There were two methods discussed in class to compute optical flow. The first method is called the Lucas-Kanade method and the second method is called here as the Image Brightness change constancy method. For this project, I have implemented and used the second method for Optical Flow estimation.

II. Theory and Methods

A. Image Brightness Constancy Equation

Motion field estimation means finding the motion vector $v = (v_x, v_y)$ from a sequence of images. To do so, we must first look at the image brightness constancy equation.

Let $I^t(x, y)$ be the intensity of a pixel (x, y) at time, t. The image brightness constancy equation just assumes that the intensity of the same pixel (x, y) remains the same at time $t + 1$. This the same as saying that

$$\frac{dI}{dt} = 0 \qquad (1)$$

This assumption or constraint will only be satisfied if 1) the motion is translational motion or 2) the illumination direction is parallel to the angular velocity for the Lambertian surface. [1]

Since I is a function of (x, y) which are themselves function of the time, t, we can expand equation (1) as,

$$\frac{dI}{dt} = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0 \qquad (2)$$

Where $\frac{\partial I}{\partial x}$ and $\frac{\partial I}{\partial y}$ represent the spatial intensity gradient, while $\frac{\partial I}{\partial t}$ represents the temporal intensity gradient. Therefore, represented in a more compact form, the image brightness constancy equation is,

$$(\nabla \mathrm{I})^{\mathrm{t}} v + I_t = 0 \qquad (3)$$

Where ∇I is called the image intensity gradient. Now that we have the image brightness constancy equation, we can use it to compute v. Note however, the equation offers no constraint on v when it is orthogonal to ∇I. Therefore, optical flow is always parallel to the image gradient.

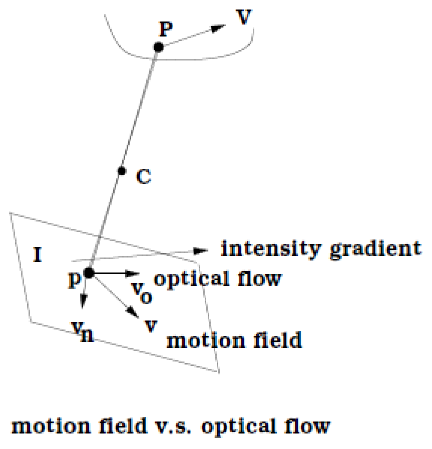

Figure 1. Motion Field & Optical Flow.(Image from [1])

B. *Optical Flow Estimation – Lucas-Kanade Method*

Equation 3 only provides us with one equation to solve for two unknowns: (v_x, v_y). Therefore, we need additional constraints to find our two unknowns. This leads us to our first method of optical flow estimation: the Lucas-Kanade method.

For a neighborhood, R, of $N \times N$ around each image point, p, with p as the center, we assume that every point in this neighborhood has the same optical flow. This means that in the neighborhood, R, (v_x, v_y) is the same for all (x, y). This is called a smoothness constraint and it provides us with the additional constraint we need to find our two unknowns. Note, however, that the smoothness constraint might not hold near the edges of moving objects). This gives the following equation,

$$\nabla^t I(x,y)v(x,y) + I_t(x,y) = 0 \tag{4}$$

where $(x,y) \in R$.

Now, $v(x,y)$ can be estimated by minimizing the following equation. The least squares solution is given in equation 6.

$$\epsilon^2 = \Sigma(\nabla^t I(x,y)v(x,y) + I_t(x,y))^2 \tag{5}$$

$$v(x,y) = (A^t A)^{-1} A^t b \tag{6}$$

Where,

$$A = \begin{bmatrix} \nabla^t I(x_1, y_1) \\ \nabla^t I(x_2, y_2) \\ \cdot \\ \cdot \\ \cdot \\ \nabla^t I(x_N, y_N) \end{bmatrix}$$

$$b = -[I_t(x_1,y_1), I_t(x_2,y_2), \ldots, I_t(x_N,y_N)]^t$$

This method of computing (v_x, v_y) has the advantage of being simple in its implementation and the fact that it uses only first order image derivatives. The first order derivatives can further be computed by first using a Gaussian smoothing operation followed by the use of a gradient operator. Also, weights can be assigned to other pixels in the neighborhood Raccording to their proximity to the center pixel p.

C. *Optical Flow Estimation - Image Brightness Change Constancy*

Instead of assuming brightness constancy "while the objects are in motion, we can assume both spatial and temporal image brightness change to be constant." Mathematically, these constraints can be written as the second derivatives of intensity being equal to 0.

$$\frac{d^2 I}{dtdx} = 0, \frac{d^2 I}{dtdy} = 0, \frac{d^2 I}{dtdt} = 0 \tag{7}$$

Applying these constraints on equation 2 gives us the these three additional equations

$$v_x I_{xx} + v_y I_{yx} + I_{tx} = 0 \tag{8}$$
$$v_x I_{xy} + v_y I_{yy} + I_{ty} = 0$$
$$v_x I_{xt} + v_y I_{yt} + I_{tt} = 0$$

Thus the final four equations we have are:

$$\begin{aligned} v_x I_x + v_y I_y + I_t &= 0 \\ v_x I_{xx} + v_y I_{yx} + I_{tx} &= 0 \\ v_x I_{xy} + v_y I_{yy} + I_{ty} &= 0 \\ v_x I_{xt} + v_y I_{yt} + I_{tt} &= 0 \end{aligned} \tag{9}$$

Now, we have four equations for two unknowns which we can solve using the following least squares problem to get (v_x, v_y).

$$\min ||Av - b||^2 \tag{10}$$

Where,

$$A = \begin{bmatrix} I_x & I_t \\ I_{xx} & I_{xt} \\ I_{yx} & I_{yt} \\ I_{tx} & I_{tt} \end{bmatrix}$$

$$b = -\begin{bmatrix} I_t \\ I_{xt} \\ I_{yt} \\ I_{tt} \end{bmatrix}$$

$$v = (A^t A)^{-1} A^t b \tag{11}$$

This is the method that I have used in this project to estimate the optical flow.

D. *Computing Image Derivatives – Cubic Facet Model*

Typically, intensity derivatives are computed as numerical approximations of continuous differentiations but we compute image derivatives analytically using the cubic facet model. The cubic facet model gives us an "analytical and continuous image intensity function that approximates the image surface at (x, y, t)." [1] This results in a more rebust and accurate estimation of image intensity derivatives because it suppresses noise through smoothing inherent to this function approximation.

For a neighborhood $k \times k \times k$, if we assume that the gray level pattern of the image sequence is "ideally a canonical 3D cubic polynomial of x, y, t,"[1] the function for the cubic facet model can be given as:

$$\begin{aligned} I(x,y,t) = {} & a_1 + a_2 x + a_3 y + a_4 t + a_5 x^2 + a_6 xy \\ & + a_7 y^2 + a_8 yt + a_9 t^2 + a_{10} xt + a_{11} x^3 + a_{12} x^2 y \\ & + a_{13} xy^2 + a_{14} y^3 + a_{15} y^2 t + a_{16} yt^2 + a_{17} t^3 \\ & + a_{18} x^2 t + a_{19} xt^2 + a_{20} xyt \end{aligned} \tag{12}$$

Where $x, y, t \in R$.

The solution for the coefficients $a = [a_1, a_2, \ldots, a_{20}]^t$ can be found through the following least squares solution:

$$\min ||Da - J||^2 \tag{13}$$

$$a = (D^t D)^{-1} D^t J \tag{14}$$

Where,

$$D = \begin{bmatrix} 1 & x_1 & y_1 & t_1 & \dots & x_1 y_1 t_1 \\ 1 & x_2 & y_1 & t_1 & \dots & x_2 y_1 t_1 \\ & \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_k & y_1 & t_1 & \dots & x_k y_1 t_1 \\ 1 & x_1 & y_2 & t_1 & \dots & x_1 y_2 t_1 \\ 1 & x_2 & y_2 & t_1 & \dots & x_2 y_2 t_1 \\ & \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_k & y_2 & t_1 & \dots & x_k y_2 t_1 \\ & \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_k & y_k & t_k & \dots & x_k y_k t_k \end{bmatrix}$$

$$J = \begin{bmatrix} I(x_1, y_1, t_1) \\ I(x_2, y_1, t_1) \\ \vdots \\ I(x_k, y_k, t_k) \end{bmatrix}$$

And $I(x_i, y_j, t_k)$ is the intensity value at $(x_i, y_j, t_k) \in R$ and R is the neighborhood volume.

Also, note that the coordinates of the pixel points in R must be considered locally. Which means that the surface created by the cubic facet model must be centered around p, the pixel being considered and that the coordinates of x, y, t will range from $-k \; to \; k$. For example, for a region of $3 \times 3 \times 3$, the coordinates of x, y, t are $-1, 0, 1 \; and -1, 0, 1, and -1,0,1$ respectively. This means that the matrix D will be the same for each point.

Finally, after computing the coefficients the image derivatives can be easily found using equation 12 and then substituted into equation 10 to get the following matrices:

$$A = \begin{bmatrix} a_2 & a_3 \\ 2a_5 & a_6 \\ a_6 & 2a_7 \\ a_{10} & a_8 \end{bmatrix} \tag{15}$$

$$b = -\begin{bmatrix} a_4 \\ a_{10} \\ a_8 \\ 2a_9 \end{bmatrix}$$

III. Experimental Procedure & Results

A. *Optical Flow Estimation Algorithm*

The input to this algorithm is a sequence of images. For our case we have 5 ($N = 5$) image frames. Next we choose a window for the region R. This window can be anything greater than a 3×3 window and we typically choose a window of $L \times L$ with $L = 5$. Now we use the following steps to estimate the optical flow on the middle image.

1) Select an image as the central frame. Usually this is the middle image so that there is an equal number of images in the sequence, before and after this image. For an image set of 5 images, this will be the third image.

2) For each pixel (excluding boundary pixels) in the middle image frame, (taken from [1])

- *Perform cubic facet model fit using equation 12 and compute the coefficients using 14.*
- *Derive the image derivatives using the coefficients and make up the matrices A and b as specified in equation 15.*
- *Compute Image flow using equation 11.*
- *Check the residual fitting error, if it is large then the fitting isn't good enough. Therefore set the optical flow that point to be 0.*
- *Check the condition of matrix A. If it is large, then set the optical flow to be zero.*
- *Check the magnitude of the optical vector. If it is very small then it means it is not that significant within a margin of error and therefore, set the flow of that point to be 0.*
- *Mark each point with an arrow indicating its flow.*

We have three thresholds here that we can control: 1) Residual Fitting Error which is just $norm(Da - J)$ or alternatively we can find the corresponding error $norm(Av - b)$ let's call this T_1. 2) Condition of Matrix A. Call this T_2. 3) Magnitude of optical vector. This has to be greater than T_3 and less than T_4.

The code for the Optical Flow algorithm is given at the end of the report. It was written in *MATLAB*. When it is run, it prompts the user to select an image set (1, 2 or 3).

All the following result figures are given on pages of their own at the end of the report to make them appear clearer.

B. *Results – Image Set I: Sphere*

The first sequence of images is that of a sphere. There are five images in this set as shown in figure 2.

The resulting optical flow is shown in figures 3 and 4. They both represent the optical flow vectors superimposed on the original middle image. The difference is in the choice of thresholds.

Both result images here clearly show the motion of the sphere. It is rotating along an axis with it's right side going into the plane of the image and left side coming out.

C. *Results – Image Set II: Checker Pattern*

The second sequence of images is that of a checkerboard pattern. There are five images in this set as shown in figure 5. Just like the sphere images, it is also a synthetic image sequence.

This image sequence was noisy and not a great optical flow estimation was obtained. So as discussed before in section II B, we can precede the optical flow estimation with a Gaussian filter to get rid of some of the noise. This was done in *MATLAB* using the `imgaussfilt(Img,Scale)` command.

The resulting optical flow is shown in figures 6 and 7. They both represent the optical flow vectors superimposed on the original middle image. The difference is in the choice of thresholds and Gaussian filter scale.

Both result images here clearly show the motion of the checker pattern. It is a translator motion in the x-direction and the pattern is moving towards the right. However, it can be still be seen that there are some vectors that are pointing in unexpected directions even after the de-noising. There could be a few reasons for that which are given in the final conclusion section.

*D. **Results – Image Set III: Moving People***

The third and final sequence of images is that of two people: a student and a profesor. The person on the left is the student and the person on the right is the professor. Again, there were five images in this sequence set as shown in figure 8. Unlike the first two sets, this is not a synthetic sequence.

This image sequence was alse a bit noisy so I've used a Gaussian filter to get rid of some of the noise. This was done in *MATLAB* using the `imgaussfilt(Img,Scale)` command.

The resulting optical flow is shown in figures 9 and 10. They both represent the optical flow vectors superimposed on the original middle image.

Although the optical flow field is not as clear here as that for the synthetic images, by looking at the resulting optical flow vectors of this image we can still see that both persons are moving. The student (person on the left) is moving towards the left and the professor (person on the right) is moving away from the camera.

Also observable here is that the length of the optical flow vectors that represent the movement of the two people vary. The length of the vectors for the person in the left are much longer than those of the professor. Since these arrows are representations of the velocity vectors: their magnitude and direction we can therefore tell that the student is moving faster than the professor.

IV. Summary & Conclusions

I achieved all the goals as laid out by the project guidelines. I have achieved good results for optical flow estimation on all three sets of image sequences.

Further work can be done to improve said algorithms and we can still play around with different parameters like making the second derivatives in equation 7 equal a constant instead of 0 or play around with the different thresholds T_1, T_2, T_3 or using a different filter.

Another thing I wanted to discuss in this section are the factors that could have affected the accuracy of the optical flow estimation for us. This was briefly mentioned in section III C. The main reason for the synthetic image is the noise inherent to the images and it cannot fully be removed using a low pass filter. Also, for the final image sequence the estimation depends a lot of the thresholds chosen so for the final sequence I've added a lower and an upper bound to the magnitude of the optical vectors. This removes arrows that are way too big or small.

One of the main problems encountered in this project was the choosing the right thresholds. This was an arbitrary process so at the start everything was computed without any thresholds in place and later by observing the results, a set of values for these thresholds was decided.

References

[1] [1] Ji, Qiang. "ECSE 6650 Lecture Notes." N.p., n.d. Web. 29 Apr. 2016.
<https://www.ecse.rpi.edu/Homepages/qji/CV/ecse6650lecture notes.html>

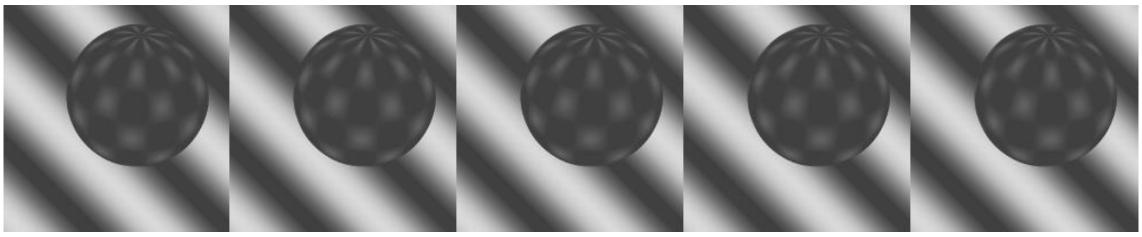

Figure 4. Image Sequence I - Sphere

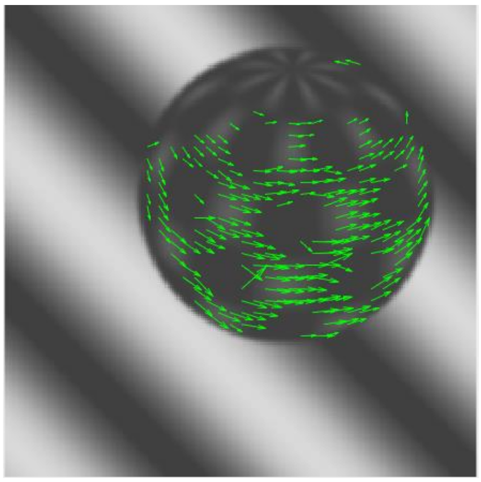

Figure 3. Optical Flow – Sphere - $T_1 = 10, T_2 = 8, T_3 = 1, T_4 = 8,$ $Filter\ Scale = 1$

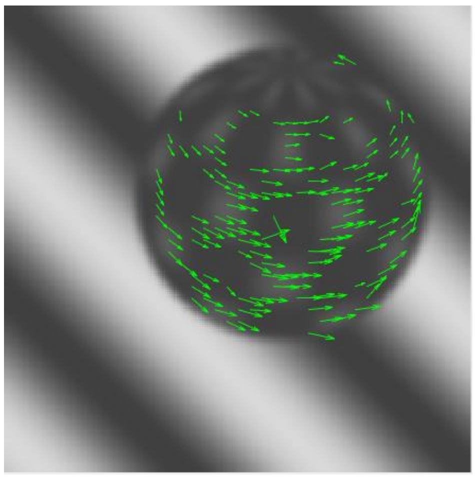

Figure 4. Optical Flow – Sphere - $T_1 = 10, T_2 = 8, T_3 = 1, T_4 = 7,$ $Filter\ Scale = 2$

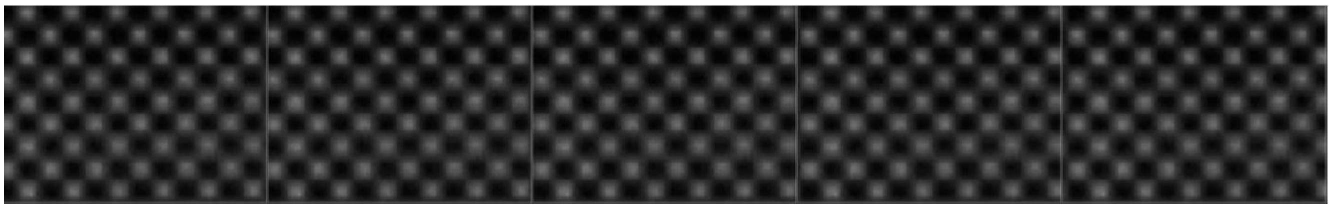

Figure 5. Image Sequence II - Checker

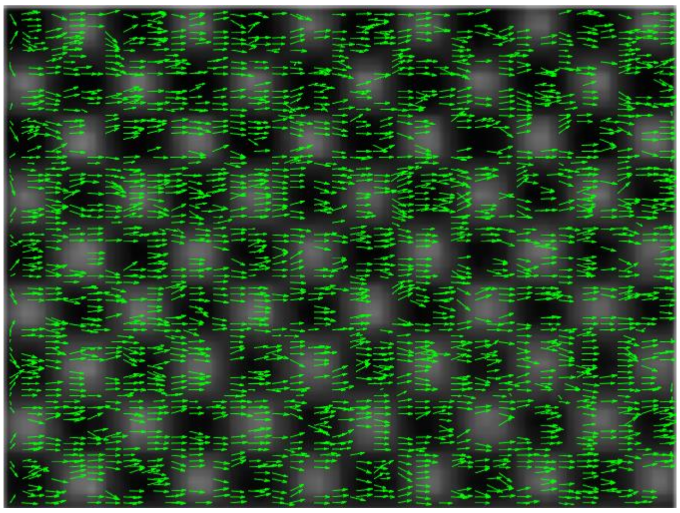

Figure 6. Optical Flow – Checker - $T_1 = 10, T_2 = 5, T_3 = 1, T_4 = 6,$ $Filter\ Scale = 2$

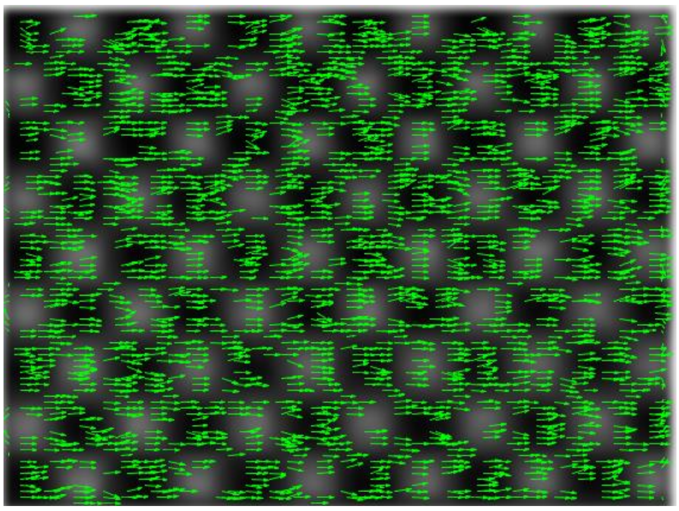

Figure 7. Optical Flow – Checker - $T_1 = 10, T_2 = 8, T_3 = 1, T_4 = 7,$ $Filter\ Scale = 4$

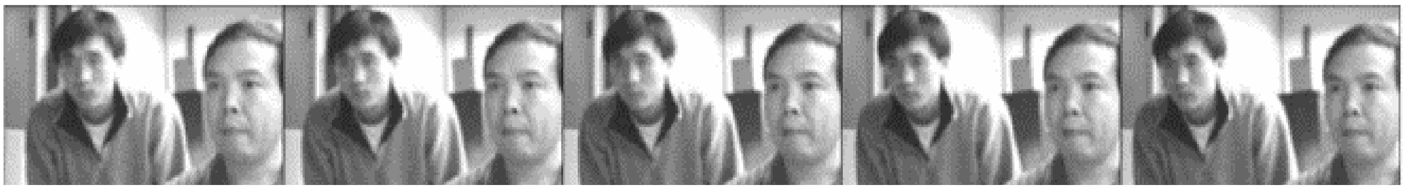
Figure 8. Image Sequence III - People

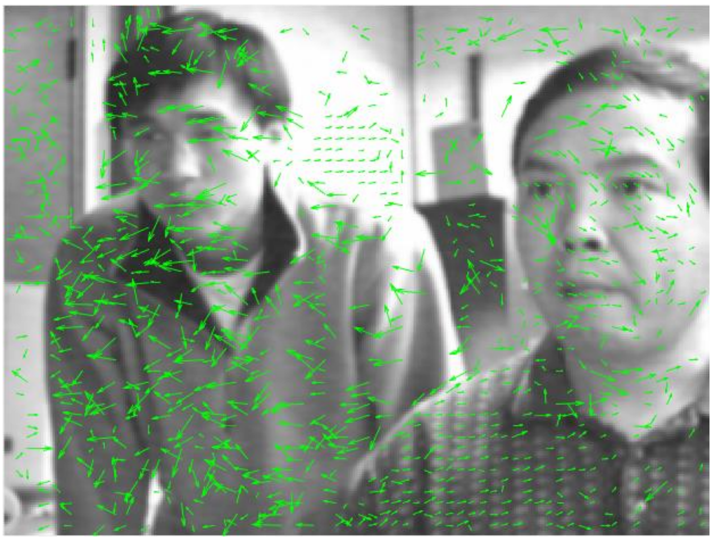
Figure 9. Optical Flow – People -
$T_1 = 10, T_2 = 5, T_3 = 1, T_4 = 6,$
$Filter\ Scale = 1$

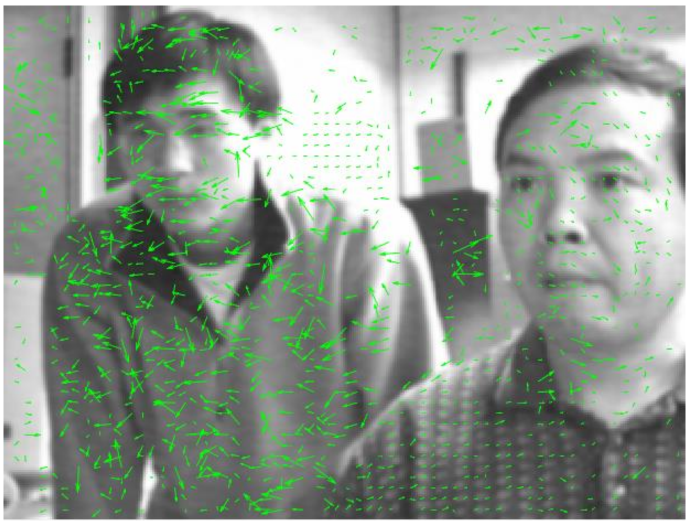
Figure 10. Optical Flow – People -
$T_1 = 10, T_2 = 5, T_3 = 1, T_4 = 10,$
$Filter\ Scale = 1$
(Note the Longer Vector lengths on the left)

CODE:

```
%% House Keeping
clc; clear; close all;
%% Read Images
I = cell(1,5);
choice1 = 0;
while (1)
    prompt1 = ' Choose image set (1, 2 or 3):    ';
    choice1 = double(input(prompt1));
    if (choice1 == 1 || choice1 ==2 || choice1 ==3)
        break;
    end
end
%Set the 3D data equal to the chosen set
if (choice1 == 1)
    for i = 1 : 5
        filename = strcat('sphere',num2str(i+1));
        filename = strcat(filename,'.pgm');
        I{i} = imread(filename);
       figure, imshow(I{i});
    end
elseif (choice1 == 2)
    for i = 1 : 5
        filename = strcat('center',num2str(i));
        filename = strcat(filename,'.jpg');
        I{i} = rgb2gray(imread(filename));
       % figure, imshow(I{i});
    end
elseif (choice1 == 3)
    for i = 1 : 5
        filename = strcat(num2str(i));
        filename = strcat(filename,'.pgm');
        I{i} = imread(filename);
       % figure, imshow(I{i});
    end
end

%digit represents time instant.
%bar represents -ive
im2bar = I{1};im1bar = I{2};im0 = I{3};im1 = I{4};im2 = I{5};
%% Gaussian Filter on Images
scalefilter = 1;
im2bar = imgaussfilt(im2bar, scalefilter);im1bar = imgaussfilt(im1bar, scalefilter);
im0 = imgaussfilt(im0, scalefilter);im1 = imgaussfilt(im1, scalefilter);im2 =
imgaussfilt(im2, scalefilter);
%% Make Images of type double
im2bar = double(im2bar);im1bar = double(im1bar);
im0 = double(im0);im1 = double(im1);im2 = double(im2);
%% Cubic Facet
[rMax,cMax] = size(im0);
%window size = X x Y x T = 3x3x5
%result = 255*ones(rMax,cMax);
%resultVX = zeros(rMax,cMax);
%resultVY = zeros(rMax,cMax);
%% Make D matrix here since it's same for all points
D =[];
for t = -2:2
    for y = -2:2
        for x = -2:2
            Dbar = [1, x, y, t, x^2,
x*y,y^2,y*t,t^2,x*t,x^3,x^2*y,x*y^2,y^3,y^2*t,y*t^2,t^3,x^2*t,x*t^2,x*y*t];
```

```
            D = [D;Dbar];
        end
    end
end
%% Compute OF (vx and vy)
result = [];
cr = [];
condition = [];
projEror = [];
VNORM = [];
for i = 1:1:rMax %row
    for j = 1:1:cMax %col
        if(i>2 && i<(rMax-1) && j > 2 && j<(cMax-1))
            J = [];
            for t = -2:2
                for y = -2:2
                    for x = -2:2
                        if t == -2
                           Jbar = im2bar(i+y,j+x);
                        end
                        if t == -1
                            Jbar = im1bar(i+y,j+x);
                        end
                        if t == 0
                            Jbar = im0(i+y,j+x);
                        end
                        if t == 1
                            Jbar = im1(i+y,j+x);
                        end
                        if t == 2
                            Jbar = im2(i+y,j+x);
                        end
                        J = [J;Jbar];
                    end
                end
            end
            a = pinv(D)*J;%D\J;
            A = [a(2),a(3);2*a(5),a(6);a(6),2*a(7);a(10),a(8)];
            condition = [condition;cond(A)];
            if (cond(A) < 8)
                b = -[a(4);a(10);a(8);2*a(9)];
                v = pinv(A)*b;%A\b;
            projEror = [projEror; norm(A*v - b)];
            VNORM = [VNORM,norm(v)];
                if norm(A*v - b) > 10
                    v = [0;0];
                end
                if norm(v) < 1
                    v = [0;0];
                end
                if norm(v) > 20
                    v = [0;0];
                end
            else
                v = [0;0];
            end
%             resultVX(i,j) = v(1,1);
%             resultVY(i,j) = v(2,1);
            result = [result ; v'];
            cr = [cr;j i];
        else
            v = [0;0];
%             resultVX(i,j) = v(1,1);
%            resultVY(i,j) = v(2,1);
            result = [result ; v'];
            cr =[cr;j i];
        end
    end
end
%[x,y] = meshgrid(1:5:cMax,1:5:rMax);
%[x,y] = meshgrid(1:cMax,1:rMax);
%result_vx = resultVX(1:5:end,1:5:end);
%result_vy = resultVX(1:5:end,1:5:end);
%result_vx = resultVX;
%result_vy = resultVX;
scale = 2;
figure;imshow(uint8(im0));hold on;
quiver(cr(:,1),cr(:,2),result(:,1),result(:,2),scale,'g');
hold off;
%16229
```