ENPM-673

Homework-Optical Flow

Perception for Autonomous Robots

UNIVERSITY OF MARYLAND

SUBMITTED BY:

ABHISHEK KATHPAL

DIRECTORY_ID: AKATHPAL

UID: 114852373

1. CONTENTS

S.NO.	TOPICS	PAGE NO.
1.)	CONTENTS	1
2.)	LUCAS KANADE IMPLEMENTAION	2-3
3.)	MATLAB OPTICAL FLOW TECHNIQUES	4
4.)	OBSERVATIONS AND RESULTS	5-8
5.)	CONCLUSION	9
6.)	REFERENCES	9

2. Lucas – Kanade Implementation

Optical Flow is a 2D vector where each vector is a displacement vector showing the movement of pixels from one frame to another. In this project, I have implemented Lucas Kanade algorithm to compute the optical flow from the given data set of grove and wooden textures. The output is plotted using quiver plots.

Lucas Kanade algorithm assumes that the objects have not displaced significantly from one frame to another and that the image consists of textured objects that exhibits different shades of gray that change smoothly.

This algorithm uses the local change in intensity i.e. gradients to compute the direction in which the object has moved with respect to previous frame.

Lucas Kanade gives error on boundaries except that it is more easier and less computational intensive in comparison with other optical flow techniques.

The implementation is based on the following brightness consistency equation written below:

$$\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = 0$$

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

The output obtained from Lucas – Kanade for Grove and Wooden texture is shown in Fig.1.

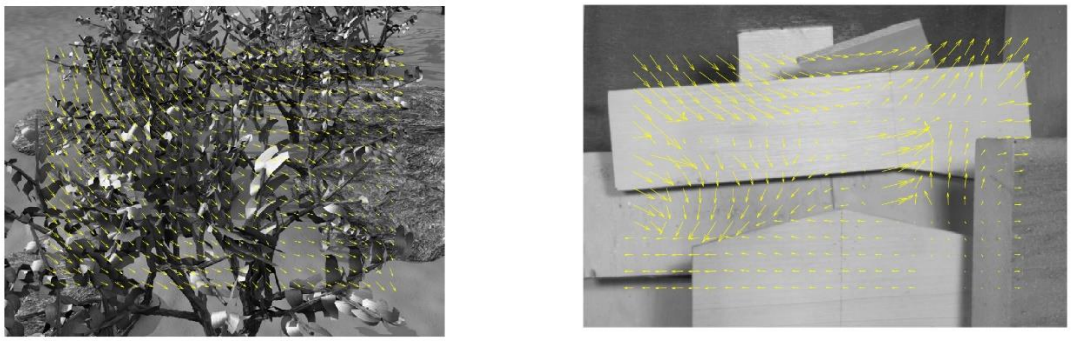

Fig.1. Output Lucas Kanade

Implementation of Lucas-Kanade is shown in Fig.2. below.

```
%% Setting parameters
% Window size
window = 45;
%Downsize factor
factor = 0.5;
```

```
%% Set Test data location
test_data = @(i) fullfile(sprintf('../Dataset/Grove/frame%02d.png',i));
% test_data = @(i) fullfile(sprintf('../Dataset/Wooden/frame%02d.png',i));
```

```
%% Set output data location
output_data = @(i) fullfile(sprintf('../Output/Grove/frame_w%d_s%d_%02d.png',window,factor,i));
% output_data = @(i) fullfile(sprintf('../Output/Wooden/frame_w%d_s%d_%02d.png',window,factor,i));

for num = 1:7
```

```
frame1 = imread(test_data(num+6));
frame2 = imread(test_data(num+7));

frame1_double = im2double(frame1);
frame2_double = im2double(frame2);

% Downsizing image by a factor
img1 = imresize(frame1_double, factor);
img2 = imresize(frame2_double, factor);
```

```
w = floor(window/2);
[f_dx,f_dy,f_dt] = partialDerivates(img1,img2);
u = zeros(size(img1));
v = zeros(size(img2));

for i = w+1:size(f_dx,1)-w
    for j = w+1:size(f_dx,2)-w
        current_fx = f_dx(i-w:i+w, j-w:j+w);
        current_fy = f_dy(i-w:i+w, j-w:j+w);
        current_ft = f_dt(i-w:i+w, j-w:j+w);

        current_fx = current_fx(:);
        current_fy = current_fy(:);
        current_ft = -current_ft(:);

        A = [current_fx current_fy];

        % A is not a square matrix, so using pseudo inverse to compute
        % velocity vectors
        vel = pinv(A)*current_ft;

        u(i,j)=vel(1);
        v(i,j)=vel(2);
    end
end
```

Fig.2. Lucas-Kanade Implementation

3. MATLAB Optical Flow Techniques

In this section, implementation of three optical flow techniques namely Lucas-Kanade, Horn-Schunck and Farneback are discussed using different MATLAB optical Flow classes. A brief description of how these classes can be used to plot velocity vectors is as follows:

3.1. Lucas-Kanade

For estimating optical flow using Lucas-Kanade Algorithm in MATLAB, opticalFlowLK class is used. In this method, to solve the optical flow constraint equation for *u* and *v*, the Lucas-Kanade method divides the original image into smaller sections and assumes a constant velocity in each section. Then, it performs a weighted least-square fit of the optical flow constraint equation to a constant model for $[u \;\; v]^T$.

3.2. Horn-Schunck

For estimating optical flow using Horn-Schunck Algorithm in MATLAB, opticalFlowHS class is used. In this method, it is assumed that optical flow is smooth over the entire image, this method computes an estimate of the velocity field, $[u \;\; v]^T$, that minimizes this equation:

$$E = \iint \left(I_x u + I_y v + I_t\right)^2 dx\, dy + a \iint \left\{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2\right\} dx\, dy$$

3.3. Farneback

For estimating optical flow using Farneback Algorithm in MATLAB, opticalFlowFarneback class is used. This method uses polynomial expansion to estimate the motion between two consecutive frames.

The implementation of all the three methods is in Optical_flow_matlab.m file. In the next section, the output from all the three techniques implemented on Grove and Wooden dataset is compared and results are discussed.

4. Observation and Results

4.1. Effect of changing window size

I have implemented the Lucas-Kanade algorithm for four different window sizes 30,45,60 and 70 on both grove dataset and wooden dataset. According to my observation, with increase in window size, the velocity vectors increase in magnitude but there are less accurate. Increasing window size helps to visualize the optical flow of the region of pixels instead of a single pixel. To get more reliable velocity vector for each pixel, it is better to keep window size small. The images of wooden and grove data set for different window sizes are shown below:

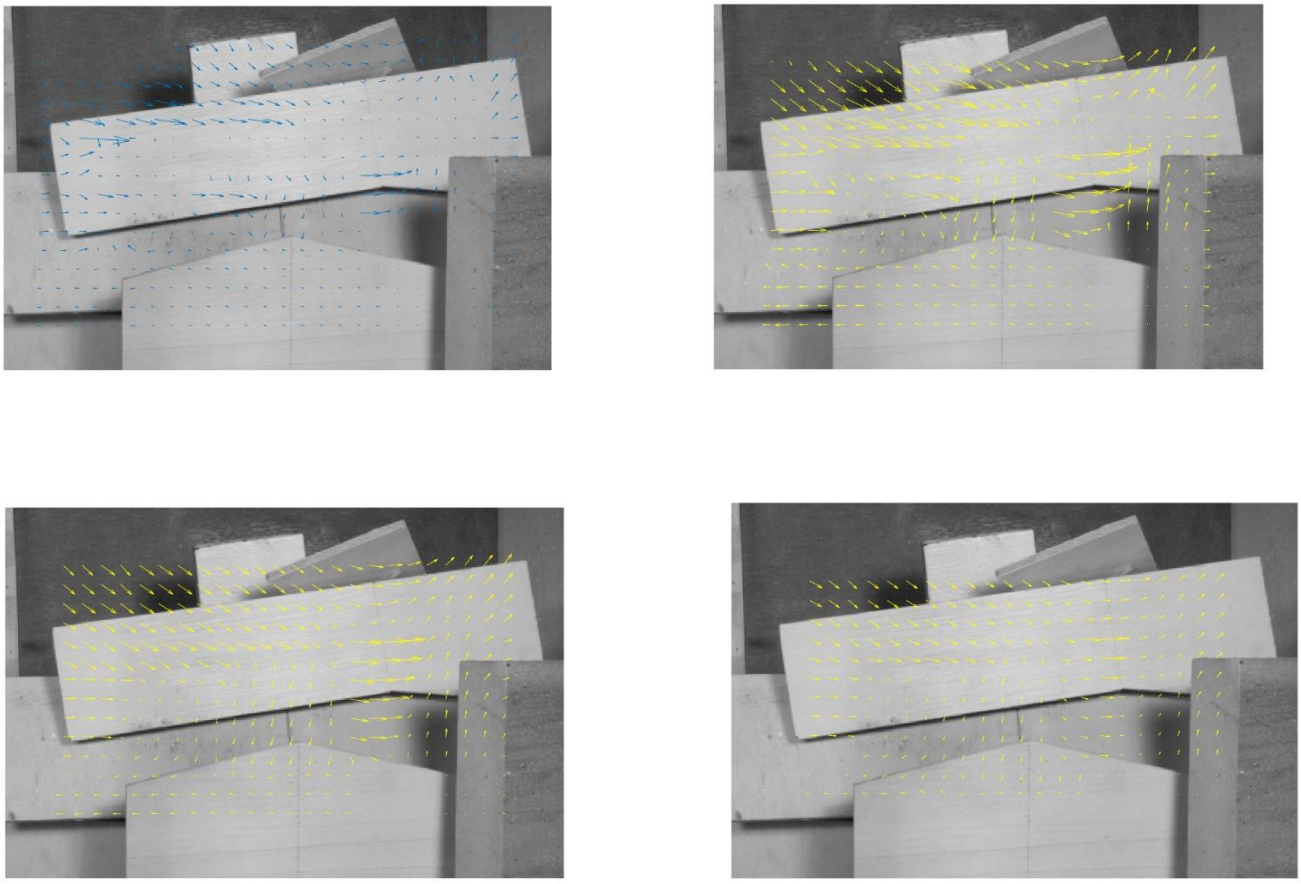

Fig. 3. Wooden Dataset for 30,45,60 and 75 window sizes

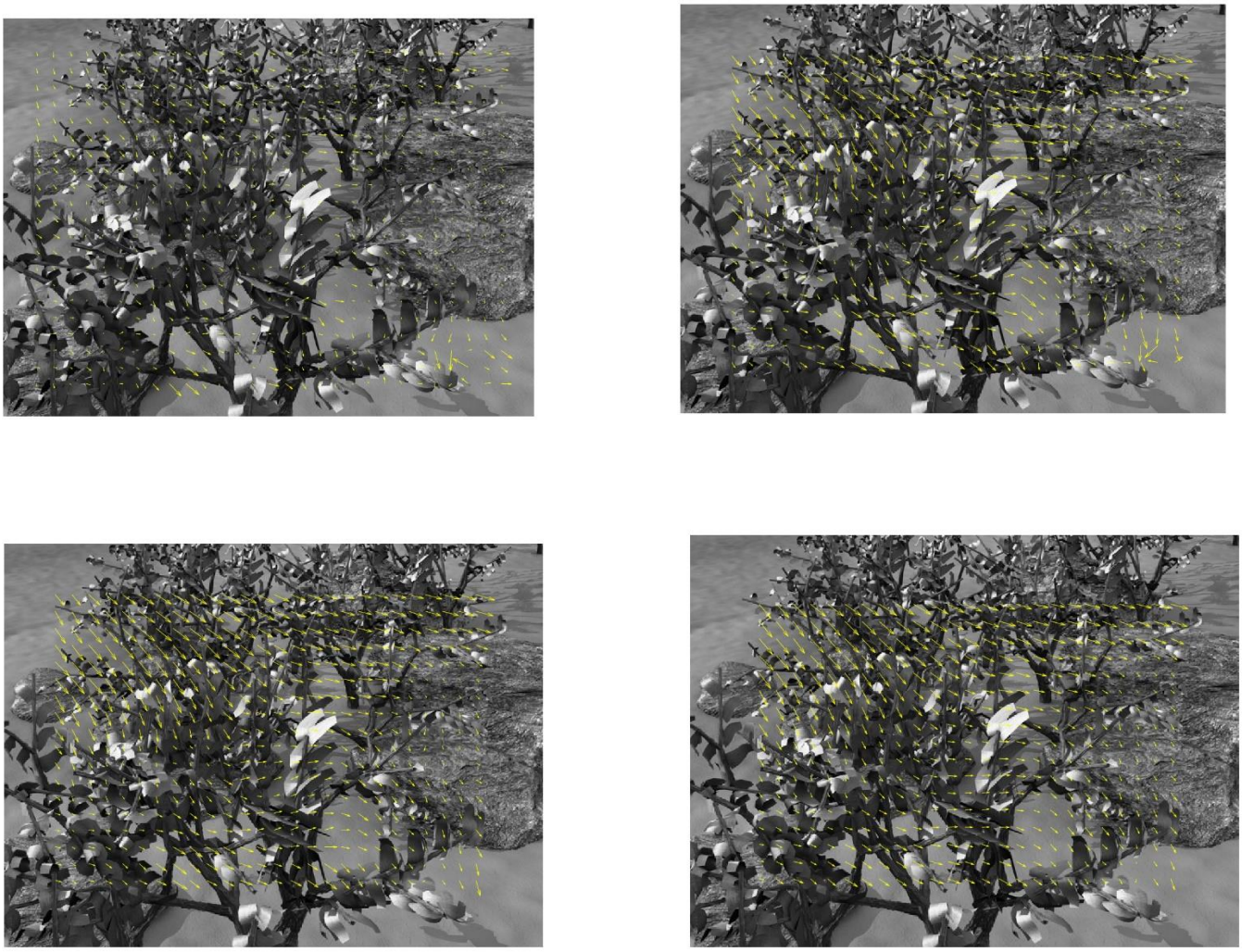

Fig. 3. Wooden Dataset for 30,45,60 and 75 window size

4.2. Effect of down-sampling

I have also tried down-sampling image to 0.5 scale factor but the outputs are almost similar as expected. For small motions, there will be no difference in the optical flow generated from down-sampling. But for large motions there might be some significant difference. The outputs for Grove dataset for different scale factor is shown in Fig.4.

Note: Velocity vectors in the output images shown in this report are not that clear. Please open the output folder to see the velocity vectors clearly.

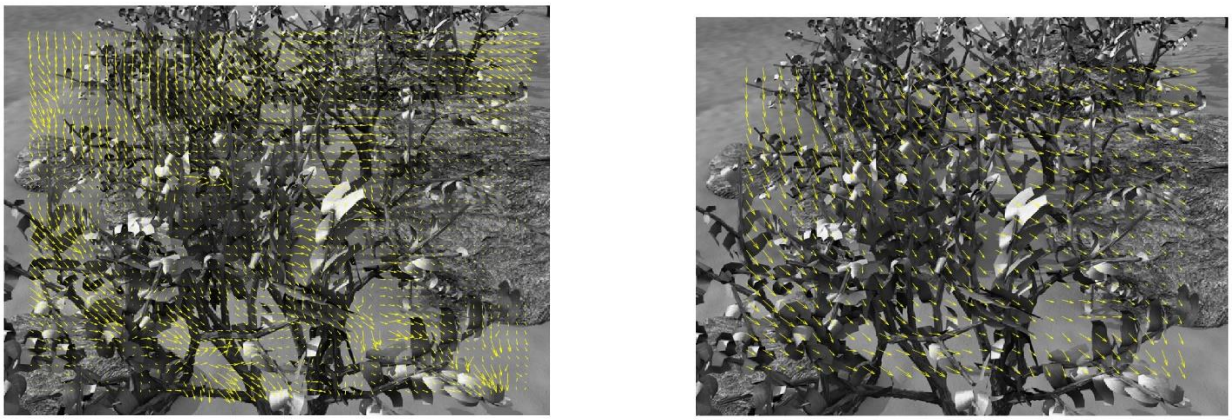

Fig. 4. Left side and Right Side (0.5 scaling)

4.3. Comparison of different techniques

The next part is to compare the MATLAB LK implementation with my own implementation of LK. This is shown in Fig. 5. The output is different because of window size used by me. If I use a window size of 10, it will be similar to MATLAB. But then the results are not good, so I used window size of 45.

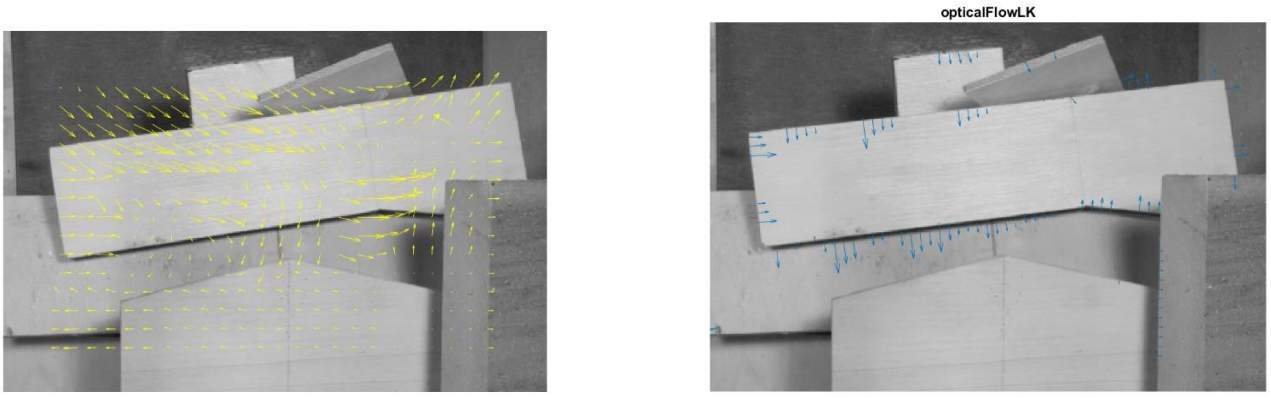

Fig. 5. Right side(MATLAB) and Left Side (My implementation)

In the figure below, I have compared the output from different optical techniques namely, Lucas-Kanade, Horn-Schunck and Farneback.

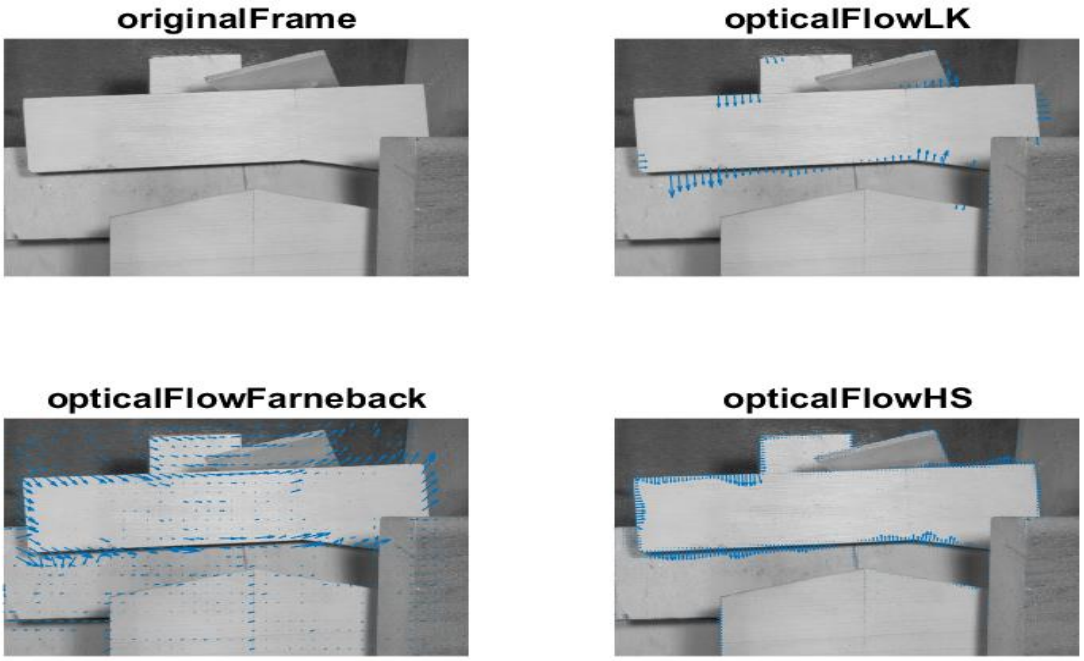

Fig. 6. MATLAB Optical Flow Wooden Output: Lucas-Kanade, Horn-Schunck and Farneback

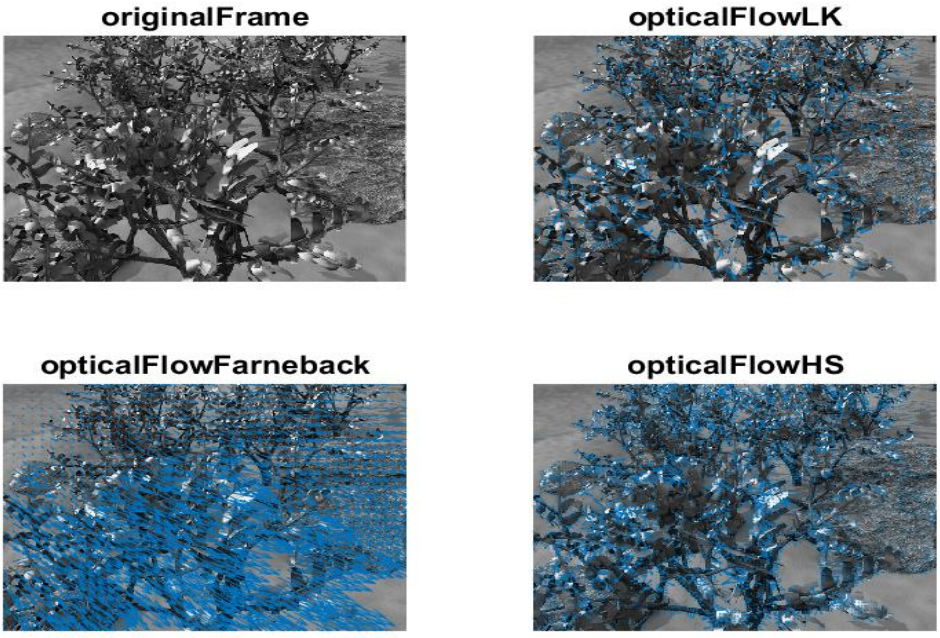

Fig. 7. MATLAB Optical Flow Grove Output: Lucas-Kanade, Horn-Schunck and Farneback

5. Conclusion

Among all the techniques, Farneback is the best technique for both textured and non-textured surfaces. Lucas-Kanade has faster execution time but less accurate. Horn-Schunck gives slightly better results for wooden dataset than Lucas Kanade. The reason for this is wooden texture is smooth and Horn-Schunck algorithm assumes smoothness in the flow over the whole image. Thus, it tries to minimize distortion in flow and prefers solutions which show more smoothness. With increased window size, Lucas-Kanade is giving better results than Horn-Schunck for Grove Dataset.

6. References

1) https://www.mathworks.com/help/vision/ref/opticalflowlk-class.html

2) https://www.mathworks.com/help/vision/ref/opticalflowfarneback-class.html

3) https://www.mathworks.com/help/vision/ref/opticalflowfarneback-class.html

4) https://www.mathworks.com/matlabcentral/fileexchange/

5) https://www.youtube.com/watch?v=5VyLAH8BhF8

6) http://vision.middlebury.edu/flow/floweval-ijcv2011.pdf