Lab #6 Post-Lab Report

pciftcioglu@sabanciuniv.edu

November 2020

1 Introduction

In this lab, we tried to do several operations with related to optical flow. Optical flow is the object's velocities and their distribution in a given sequence of images and to measure optical flow we estimate the change in apparent velocities which results in motion we observe in the scene relative to us as a viewer.

All of the operations have been performed on the gray-scale sequence of images (i.e. video) and loaded as **Seq** variables (see Appendix lab6OFMain.m). After defining a window size parameter and a threshold value for eigen values, each and all the images ("Current Image") in the sequence are sent to the function lab6OF.m with the image preceding ("Previous Image") them in the sequence. Additionally, function has been utilized in a loop to see the results clearly while the images creating a moving visual. Initially, the algorithm for optical flow function that we have done in the lab will be explained with the methods used and accordingly, and further trials with using different window size parameter and different smoothing operations will be discussed with my own results.

2 In Lab Implementation

2.1 Optical Flow

The function lab6OF.m is used to utilize this algorithm and two images (one preceding the other) has been given to the function as inputs with the window size parameter (k) and a threshold value. First, we smoothened both the current and the previous image with using a built-in function imboxfilt() with sending the double() format of the image since we want it to return as double as we will use in our further computations. imboxfilt() utilizes the smoothing operation with Box Filtering as we did in Lab 1.

Then we calculated the smoothened version of images' spatial gradients by using Prewitt Filters and convolving (in 2D) them with x_filter and y_filter with the previous (ImPrev_Smoothened) and current (ImCurr_Smoothened) images respectively (return results are Ix and Iy). Then we found the temporal gradient (It) which is calculated as subtracting the current image from the previous image's pixel values. Then we initialized matrices with zeros as the same size of the current image as Vy and Vx for velocities in both x and y direction. We also initialized the G and b values where we calculate their indices in two nested loops sliding in the window of size k. k is taken as an input and was k = 20 in our case.

After we calculated the element of G and b as shown in the lab document, we calculated the eigen values of G (lambda1 and lambda2) and used threshold method where we compare the minimum of the eigenvalues to the threshold (Threshold = 2*10$\hat{6}$ in our case). And if the min(lambda1, lambda2) is bigger than the threshold we calculated the the variable u with using the formula from the document again such that they correspond to the indices of velocity vectors of Vx and Vy. After all, we did the visualization part and the resulting image sequence can be seen from the Figure 1.

Figure 1: Optical Flow on Traffic

As it can be seen from the image, red arrows represents the direction of the velocity and they are in line with the direction of the traffic flows. We also tried another example where the velocities might not be that apparent as in the traffic example. However, the algorithm was again pretty successful to detect the way of the motion as it can be seen from the Figure 2.

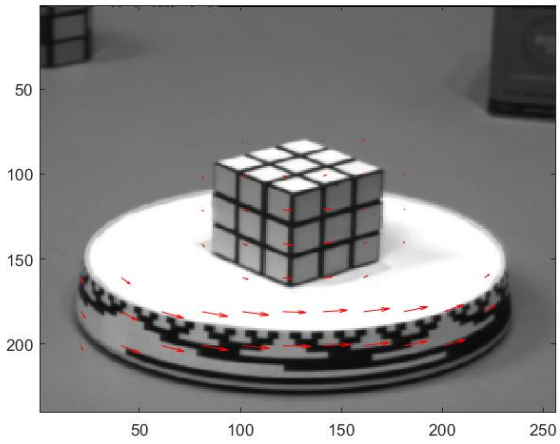


Figure 2: Optical Flow on Rubic Turning on a Table

Even though its impossible to see the actual motion on the paper, the rubic cube was actually turning in the direction on the table with inline the directions of the arrows point. So its possible to state that again the motion has been detected correctly.

Here the following are some other examples provided in the lab with all used the same threshold value and the window size parameter and they all sort of ended up correctly in terms of the motion and the direction that arrows point.

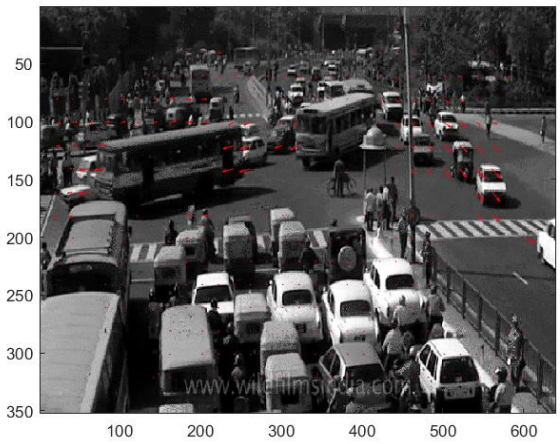


Figure 3: Optical Flow on Rubic Turning on a Table

Figure 4: Optical Flow on Traffic with Some Cars Crossing the Road

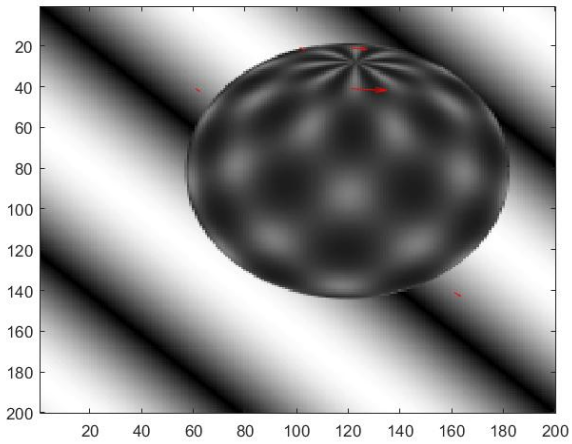


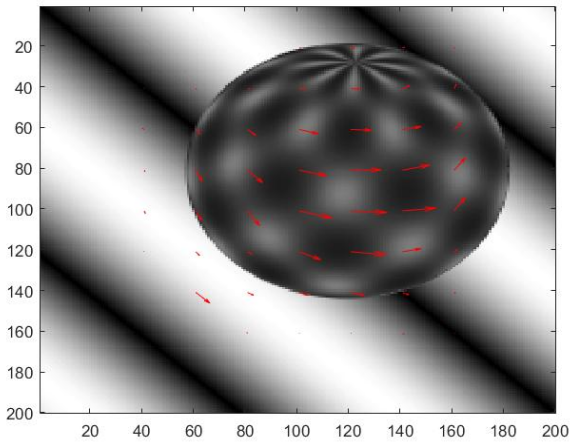
Figure 5: Optical Flow on Traffic with Cars in the Same Direction

Figure 6: Optical Flow on Rubic Turning on a Table

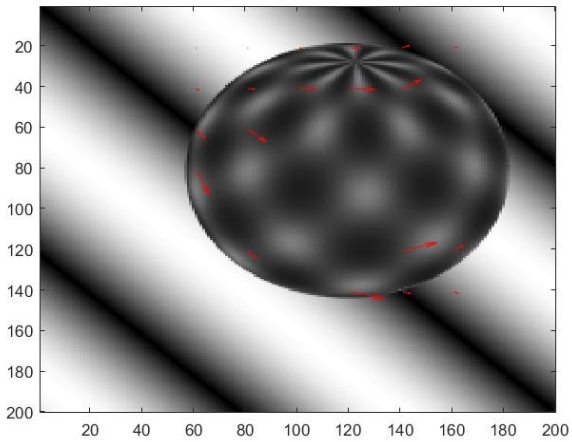
It is also important to indicate that the images that the Figures represents actually sometimes shows less arrows that it resulted so its better to check he actual results at that point. But even then, although some image sequences resulted in a pretty accurate and good motion detection such as in the examples of traffic as in the Figures 3, 4 and 6; some of them very a bit inadequate. So I will be exploring different threshold values or different window size parameters for some of the examples to see whether if its attainable to have better results.

3 Post Lab Experiments

First I tried lower and higher threshold values to observe the difference in terms of the accuracy of the arrows such that they might be more on some areas and clearer.



(a) Optical Flow with Threshold = 20



(b) Optical Flow with Threshold = 5*10$\hat{5}$

Figure 7: Optical Flow on Sphere on a Turning Table

When I tried higher threshold values, I observed nearly no arrows on the resulting sequence and the reason behind it that the algorithm might have failed to find eigen values that are higher than the threshold that I picked so it could not calculate and velocities. Thus, I did not include them but tried two different lower threshold values instead. As can be seen from the Figure 7 that the lowest threshold I tried (7a actually detected motion all around the sphere which was more accurate than the initial result however it also detected some in the background which was inaccurate. The other threshold values I tried were more accurate in terms of they did not find a motion outside the sphere but but still the velocities were underrepresented as in can be seen from the Figure 7b. Thus it might be possible to say that the optimum threshold value is closer to within range [500000,200000].

Additionally I tried different window size parameter to see its effect on the detection of motion on the previous experiment of 3 because the motion was actually underrepresented such that it was nearly invisible too see some cars moving based on the arrows (not for the cars that are waiting but the cars that are crossing the road).

(a) Optical Flow with k = 5

(b) Optical Flow with k = 30

Figure 8: Optical Flow on Traffic with Some Cars Crossing the Road

When I used lower values for the window size parameter, as can be seen from the Figure 8a, the optical flow became pretty hard to observe such as the arrows on the cars that are crossing the road is reduced. On the other hand, when I increased the window size parameter up until k = 50, the arrows got denser and the amount increased but the accuracy went lower. Thus, it is possible to indicate that the window size parameter k=20 gives the best results compared to others, as can be seen from the Figure 3.

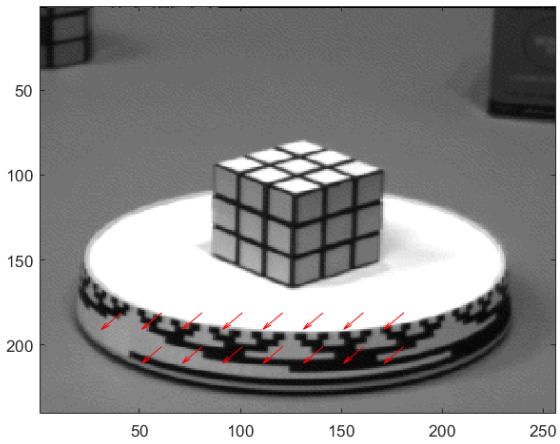


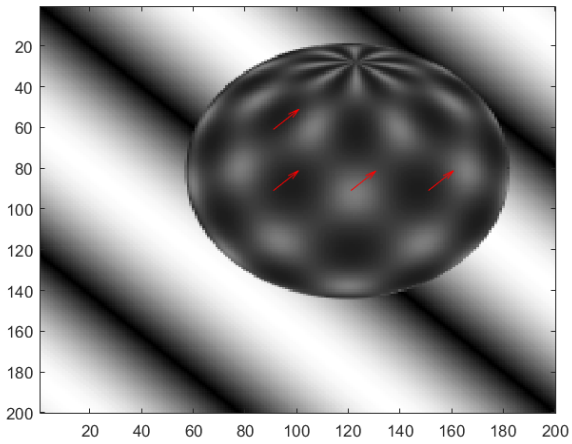
Figure 9: Optical Flow with k = 30 and Threshold = 50000, Box Filtering

After trying out different threshold values and window parameters, I wanted to see the affect of different filters for calculating spatial gradients in the first phase of the algorithm with the determined relatively optimal parameter values. As can be seen in the Figure 9, when I directly used the built in function imboxfilt(), the arrows appeared to be on the wrong side. This created an automatic answer in my brain of the question that calculation method matters in terms of how me find the temporal gradient as **It = ImCurr_Smoothened - ImPrev_Smoothened** . Thus when I changed the order of the images, it resolved itself. However, even the direction is wrong, the vector itself was also wrong so I kept on trying.

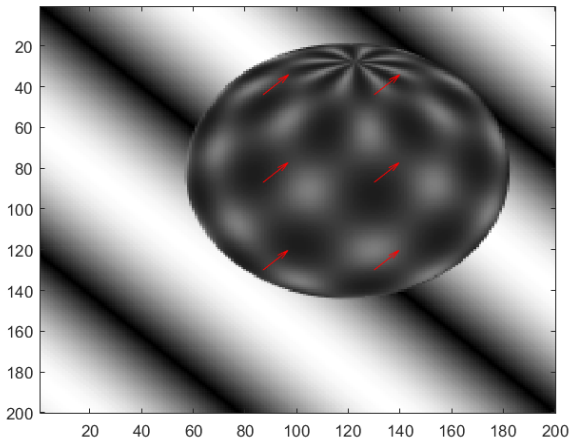
And as can be seen from the Figure 10 that the direction resulted in the correct orientation but the arrows were pointing more up than they should. I tried with different parameters but it also resulted in arrows disappear. The reason behind I trying two different threshold values as 30 and 40 is that I wanted to also see the effect of window size parameter thinking that maybe its being big enough resulted in lack of global control while calculating the G matrix. However, while the result was not getting any better, the arrows density (frequency) appearing on the image got higher as I increased the window parameter. And this time it was easier to observe since there were not that many arrows. Thus it can be clearly seen from the Figures 10a and 10b that as the k values increases the direction vectors of the velocity appears more.

The last trial I wanted to observed the results was related to the wrong results I have received on the last trials. So after that, I thought it might be the case that complexity of the images and the way pixel values (in terms of gray-scale channel) distributed among the buffer might be one factor that was affecting it. Thus I tried another image where the velocities are more in line and easier to compute because of the radical and apparent changes on the local pixels.

As a result, which was pretty surprising for me, it worked way too accurate on the image on Figure 11a as it can be seen from the velocity arrows. Thus, I also tried Gaussian filtering as a smoothing method (to calculate gradients) on this image and as can be seen from the Figure 11b compared to Figure 11a, they resulted pretty similar. It is also important to mention that I used the same threshold values that we obtained from the previous results because they were still giving the optimum results and when the window size parameter decreased under the k = 30, the result showed no vectors.



(a) Optical Flow with k = 30 and Threshold = 50000



(b) Optical Flow with k = 40 and Threshold = 50000

Figure 10: Optical Flow on sphere with Box Filtering

(a) Box Filtering

(b) Gaussian Filtering

Figure 11: Optical Flow on Traffic with k = 30 and Threshold = 200000

Finally, I left the built in functions use the internal appropriate automatic parameters (such as the sigma value and the window size) of the Gaussian and Box Filtering so it is possible to indicate that with better parameter testing and calculations it might be better to observe better results. But in terms of accuracy, event hough Box filtering resulted in less velocity indicator vectors, in terms of accuracy it still performed better than the Gaussian filter under the same parameters on the same image.

And its impossible to know the internal calculations based on the smoothing functions used because the direction of the optical flow might result wrong because of the calculation of the temporal gradient.

In addition, as looked in the final results, it might be said that the initial methods we used resulted better compared to further ones (i.e. Prewitt resulted better than Box and Gaussian) but it is still possible to say that the optimal parameters and the best method applicable vary between different images so there needs to still be done more observation in terms of determining the best performing algorithm for each unique image sequence.

Appendix

3.1 Lab6OFMain.m

```
        clear all; close all; clc;
        % Load the files given in SUcourse as Seq variable
        load('traffic.mat');
        load('cars1.mat');
        load('cars2.mat');
        load('rubic.mat');
        load('sphere.mat');
        load('taxi.mat');

        Seq = traffic;    % pick the image to use
        [row,col,num]=size(Seq);
        % Define k and Threshold
        k = 20; % Window size parameter
        Threshold = 2*10^6;   % Threshold for eigenvalues such that they should be higher than
    threshold

        while true
            for j=2:1:num
            ImPrev = Seq(:,:,j-1);  % Previous image in the sequence
            ImCurr = Seq(:,:,j);    % Current image in the sequence
            lab6OF(ImPrev,ImCurr,k,Threshold);
            %pause(0.1); % When you comment out this -> add drawnow to the function
            end
        end

```

3.2 Lab6OF.m

```
        function lab6OF(ImPrev,ImCurr,k,Threshold)

% Smooth the input images using a Box filter
ImPrev_Smoothened = imboxfilt(double(ImPrev),3);
ImCurr_Smoothened = imboxfilt(double(ImCurr));

% Calculate spatial gradients (Ix, Iy) using Prewitt filter
x_filter = [ -1 0 1 ; -1 0 1; -1 0 1 ];
y_filter =  [-1 -1 -1 ; 0 0 0; 1 1 1];
% Conv2d with filters on current image
Ix = conv2(ImPrev_Smoothened,x_filter,'same');
Iy = conv2(ImCurr_Smoothened,y_filter,'same');
%Ix = imgaussfilt(ImPrev_Smoothened);
%Iy = imgaussfilt(ImCurr_Smoothened);
%Ix = imboxfilt(ImPrev_Smoothened);
%Iy = imboxfilt(ImCurr_Smoothened);


% Calculate temporal (It) gradient (Previous Image will be only used for this)
It =   ImPrev_Smoothened - ImCurr_Smoothened;

[ydim,xdim] = size(ImCurr);
Vx = zeros(ydim,xdim);
Vy = zeros(ydim,xdim);
G = zeros(2,2);
b = zeros(2,1);

for x=k+1:k:xdim-k-1
    for y=k+1:k:ydim-k-1
        % Calculate the elements of G and b
        G(1,1)= sum(sum( Ix(y-k:y+k,x-k:x+k).^2 ));
        G(1,2)= sum(sum(Ix(y-k:y+k,x-k:x+k).*Iy(y-k:y+k,x-k:x+k)));
        G(2,1)= sum(sum(Ix(y-k:y+k,x-k:x+k).*Iy(y-k:y+k,x-k:x+k)));
        G(2,2)= sum(sum( Iy(y-k:y+k,x-k:x+k).^2 ));
```

```
35
36          b(1,1)= sum(sum(Ix(y-k:y+k,x-k:x+k).*It(y-k:y+k,x-k:x+k)));
37          b(2,1)= sum(sum(Iy(y-k:y+k,x-k:x+k).*It(y-k:y+k,x-k:x+k)));
38
39          e = eig(G);
40          lambda1 = e(1);
41          lambda2 = e(2);
42          if (min(lambda1, lambda2) > Threshold)
43              % Calculate u
44              u = -1*(inv(G)* b);
45
46              Vx(y,x)=u(1);
47              Vy(y,x)=u(2);
48          end
49      end
50  end
51  cla reset;
52  imagesc(ImPrev); hold on;
53  [xramp,yramp] = meshgrid(1:1:xdim,1:1:ydim);
54  quiver(xramp,yramp,Vx,Vy,10,'r');
55  colormap gray;  drawnow; %drawnow is just flushing mechanism to make sure all drawings are done
        before you move on
56
57  end
58
```