EE 569 3D Vision: Post-Lab Report 6

Furkan Gül, Student ID: 28795

December 17, 2020

1 Experiments in Lab

This first part of the report is an introductory step to understand scripts, and functions implemented in Lab deeply.

Given a sequence of images or video, objects that appear in the image have some velocities. Optical flow extracts the distribution of these velocities of objects in an image. In Lab, the optical flow topic is discussed and input video clips are processed to calculate the optical flow. There are one script and one function for calculating the optical flow. Before diving into the optical flow function discussed in Lab, it will be better to mention the main script that calls the required optical flow function benefited in Lab. It is named "lab6OFMain", which can be found in the Appendix. In this main script, given video clips or a sequence of images for Lab are loaded into the workspace of Matlab. After specifying window size parameter, k, and a threshold for eigenvalues of G matrix, every two subsequent frames in this sequence of images will be given to the optical flow function together with window size parameter and threshold as inputs. This function is named "lab6OF" and the optical flow algorithm will be addressed in that function. It will return sequence of images with calculated flow velocities on them. This function can be found in the Appendix. Firstly, To avoid arithmetic errors due to using image matrices of uint8 data type, which is the default option for image matrices in Matlab, this optical flow function will convert data types of two subsequent frames of image matrices to double format at the beginning. These two frames of images can be considered as the previous time frame image and the current time frame image. Then, to smoothen these two images by box filtering method, built-in Matlab function imboxfilt is used with the window size parameter of 3. Secondly, these two smoothened images will be differentiated by using Prewitt filter. For this derivative operation, after Prewitt derivative filters in x and y directions are defined, another built-in function conv2 is called to convolve the smoothened version of the previous time frame image with these two derivative filters. Thirdly, temporal gradient is calculating by taking the difference between the smoothened previous time frame image and the smoothened current time frame image. To calculate the optical flow velocity vector, G matrix and b vector must be calculated by using the equations given in the class and Lab. Optical flow velocities in x and y directions, G matrix, and b vector are all initialized as zero matrices and zero vector respectively. Then, by sliding over each pixel point in images, G matrix and b vector will be calculated at each point by

using window-sized spatial gradients in x and y directions and temporal gradient. Then, the minimum eigenvalue of G matrix will be thresholded with the given threshold value. If it exceeds the threshold value, the corresponding point will be a candidate feature point. At this feature point, the optical flow velocity vector, u, will be obtained using Equation 1 below. Finally, using imagesc and meshgrid built-in functions, the resulting image will be displayed with scaled colors. Since there is a for loop in the main script, this optical flow function will be called at each two subsequent time frames of images and the output of this function will be the input sequence of images together with the corresponding optical flow velocities on them. For the illustration in Lab, two different sequences of images are used which include the traffic image sequence and the rubic image sequence. To better appreciate the role of the optical flow illustration, if statement is added inside the for loop of the main script. By putting a break-point in the line of continue command in if statement of the main script, the Matlab figure will show the first image frame, the middle image frame, and the final image frame in the image sequence as an output. Window size parameter, k, and the threshold value are kept constant as 20 and $2 * 10^3$. The resulting optical flow of these two input images at the specified three time steps (the beginning, the middle and the end) can be seen in Figures 1 and 2. On the other hand, the effect of using different threshold values, window size parameters, and smoothing methods will be discussed in the next section.

$$u = -G^{-1}b \tag{1}$$

Figure 1: Display of the Optical Flow of Traffic Video Clip at time steps specified in the bottom-left corners on each three image frames

2 Results & Discussions

In Lab, six different .mat files are provided including the video clips to calculate the optical flows by using the implemented algorithm during Lab. Two of them, traffic and rubic, are used during the Lab session. In this report, three of them which are taxi, cars1, and sphere video clips, will be used to analyze the effect of using different window sizes, threshold values, and smoothing filters on the optical flow.

Note that all the resulting optical flow figures consist of three image frames which are the beginning, the middle, and the end of the corresponding input image sequence.

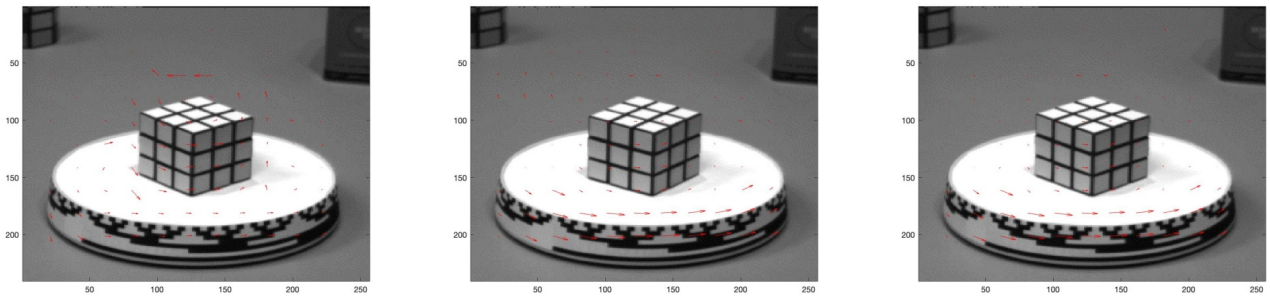


Figure 2: Display of the Optical Flow of Rubic Video Clip at time steps of 2, 11, and 20 (from left to right)

To see only the effect of window size, k; smoothing filter and threshold value will be constant as box filtering and $2 * 10^3$ respectively. For this experiment, a taxi image sequence will be used with window size parameter of 10, 20, and 30. All three resulting optical flow images for each of these window size parameters can be seen in Figures 3, 4, and 5 respectively. As it can be seen from these three Figures, as the windows size parameter, k, increases, the total number of feature points for the optical flow detection becomes less. The reason for that deduction arises from the fact that when k is increased too much (equivalently window size is increased), the original input image will be not be slid over in an intense and detailed manner. Thus, many important feature points for pointing the optical flow will become less important and less valuable when k increases. However, the magnitudes of the optical flows/velocities increase as k increases.

Figure 3: Optical Flow of Taxi Image Sequence with window size parameter of 10, Box Filtering and threshold value of $2 * 10^3$.

To observe the effect of smoothing filter, Gaussian filtering is applied by using built-in imagaussfilt function instead of Box filtering. The resulting change in the optical flow vectors of the taxi image sequence can be seen in Figures 6 and 7. When the optical flow vectors in Figures 3 and 6 are compared (where the only difference is the type of smoothing filtering), it is obvious that the optical flow vectors where Gaussian filtering is applied are less sensitive to noise since it removes local noise better than Box filtering. That's why candidate feature points for the optical flow detection seem more compact and representative in Gaussian smoothing applied cases.

Figure 4: Optical Flow of Taxi Image Sequence with window size parameter of 20, Box Filtering and threshold value of $2 * 10^3$.

Figure 5: Optical Flow of Taxi Image Sequence with window size parameter of 30, Box Filtering and threshold value of $2 * 10^3$.

Figure 6: Optical Flow of Taxi Image Sequence with window size parameter of 10, Gaussian Filtering and threshold value of $2 * 10^3$.

Figure 7: Optical Flow of Taxi Image Sequence with window size parameter of 20, Gaussian Filtering and threshold value of $2 * 10^3$.

To get a better understanding of how the threshold value affects the performance of detecting the optical flow vectors, the cars1 image sequence is processed with window size parameter of 20 and Gaussian filtering. But, two different threshold values, $2*10^1$ and $2*10^5$, will be gıven to the optical flow function. The resulting optical flow vectors in cars1 image sequence can be seen in Figures 8 and 9. As it is noticeable from these two Figures, due to the increase in threshold value, fewer feature points will become candidates for detecting the optical flow vectors. That's why there are less optical flow vectors in Figure 9 compared to the optical flow vectors in Figure 8 where the threshold is smaller than that in Figure 9.

Figure 8: Optical Flow of Cars1 Image Sequence with window size parameter of 20, Gaussian Filtering and threshold value of $2 * 10^1$.

The last image sequence which is a sphere will be used as a showcase study to tune all hyperparameter learned to get better optical flow vectors. Hyperparameters are tuned by hand. But, there is a ciritical issue for this sphere case. Since the sphere image sequence has more noise than the other two image sequences, it is wiser to be careful while choosing the smoothing filter type. For the sphere case in this report, k is chosen as 20, the threshold value is $1 * 10^5$, and the smoothing filter is Gaussian. The resulting Matlab plot for this optical flow vectors can be seen in Figure 10.

Figure 9: Optical Flow of Cars1 Image Sequence with window size parameter of 20, Gaussian Filtering and threshold value of $2 * 10^5$.

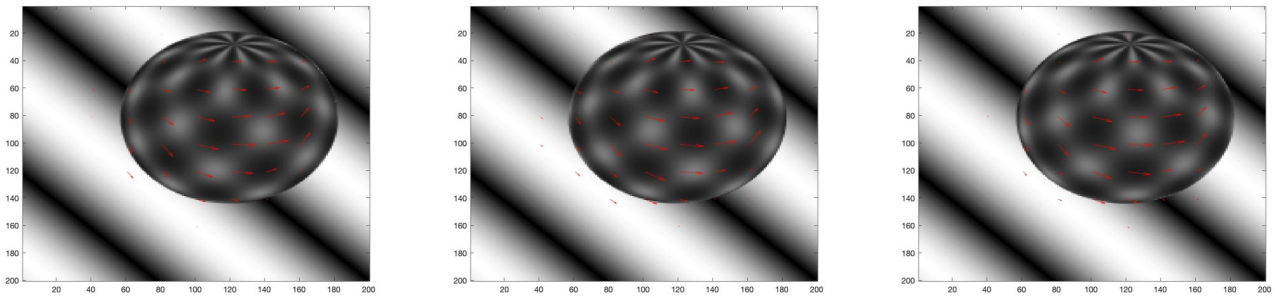


Figure 10: Optical Flow of Sphere Image Sequence with window size parameter of 20, Gaussian Filtering and threshold value of $1 * 10^5$. All three parameters optimized by hand.

3 Appendix

All the below codes and their corresponding outputs were obtained as latex output format without any manual intervention by using the "Publish" Method in Matlab.

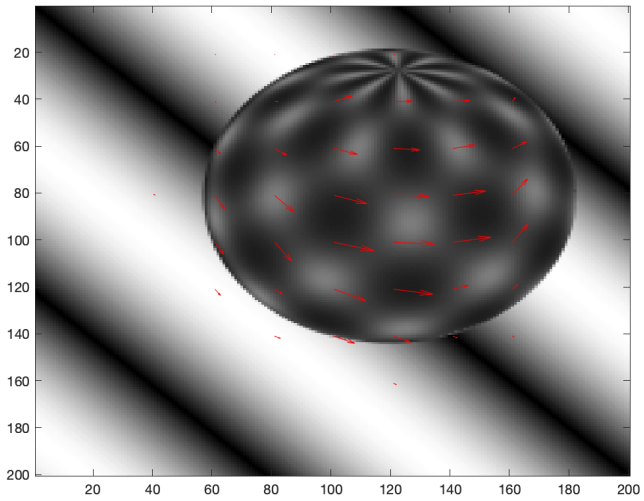
lab6OFMain Script

```
clear; close all; clc;

% Load the mat files given in SUcourse as Seq variable
load("traffic.mat");
load("rubic.mat");
load("taxi.mat");
load("cars1.mat");
load("cars2.mat");
load("sphere.mat");

Seq = sphere;
[row,col,num]=size(Seq);
% Define k(window size parameter) and Threshold for eigenvalues
k=20;
%Threshold = 2*10^6;
Threshold = 1*10^5;

for j=2:1:num
    ImPrev = Seq(:,:,j-1);
    ImCurr = Seq(:,:,j);
    lab6OF(ImPrev,ImCurr,k,Threshold);
    % For taking three image frames in sequence
    if j == 2 || j == floor(num/2) || j == num
        continue
    end
    %pause(0.1);
end
```



lab6OF Function

```
function lab6OF(ImPrev,ImCurr,k,Threshold)
% Turn them into double format to avoid algebraic errors
ImPrev = double(ImPrev);
ImCurr = double(ImCurr);
% Smooth the input images using a built in Box filter, size of 2 or 3
%ImPrev_smth = imboxfilt(ImPrev,3);
%ImCurr_smth = imboxfilt(ImCurr,3);

% Smooth the input images using a built in Gaussian filter, size of 2 or 3
ImPrev_smth = imgaussfilt(ImPrev,3);
ImCurr_smth = imgaussfilt(ImCurr,3);


% Calculate spatial gradients (Ix, Iy) using Prewitt filter and use conv2

% Define prewitt filters for x and y
x_Filter = [-1 0 1;-1 0 1; -1 0 1];
y_Filter = [-1 -1 -1; 0 0 0 ; 1 1 1];

Ix = conv2(ImPrev_smth, x_Filter, "same");
Iy = conv2(ImPrev_smth, y_Filter, "same");

% Calculate temporal (It) gradient
It=(ImPrev_smth -ImCurr_smth);
```

```
[ydim,xdim] = size(ImCurr);
% Velocities
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
        % Find window sized gradient in x and y directions

        Ix_ = Ix(y-k:y+k, x-k:x+k);

        Iy_ = Iy(y-k:y+k, x-k:x+k);

        It_ = It(y-k:y+k, x-k:x+k);

        % Find elements of G matrix
        g11 = sum(sum(Ix_.^2));
        g12 = sum(sum(Ix_.*Iy_));
        g21 = g12;
        g22 = sum(sum(Iy_.^2));
        % Find elements _of b matrix
        b11 = sum(sum(Ix_.*It_));
        b21 = sum(sum(Iy_.*It_));
        % Form G matrix
        G(1,1) = g11;
        G(1,2) = g12;
        G(2,1) = g21;
        G(2,2) = g22;
        % Form b matrix
        b(1,1) = b11;
        b(2,1) = b21;
        % Finding eigenvalues and thresholding
        [~, eigval] =eig(G);
        if min(eigval(1,1), eigval(2,2))>Threshold
            % Calculate u
            u = -inv(G)*b;
            Vx(y,x)=u(1,1);
            Vy(y,x)=u(2,1);
        end
    end
end
```

```
% Displaying part using imagesc and meshgrid with scaled colors
cla reset;
imagesc(ImPrev); hold on;
[xramp,yramp] = meshgrid(1:1:xdim,1:1:ydim);
quiver(xramp,yramp,Vx,Vy,10,"r");
colormap gray;
drawnow;
end
```