EE 417 - Computer Vision
Post Lab Report

Optical Flow

Name: Çiğdem Ceyda Düzgeç

Student ID: 23928

Date: 18.11.2020

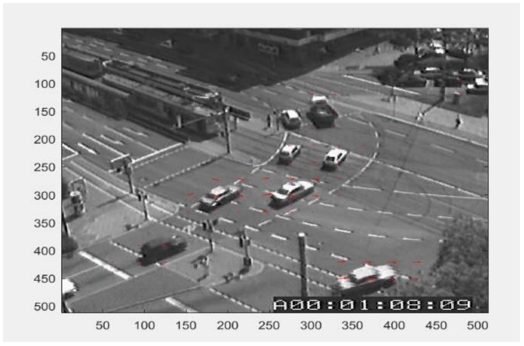
Optical Flow (lab6OFMain.m)

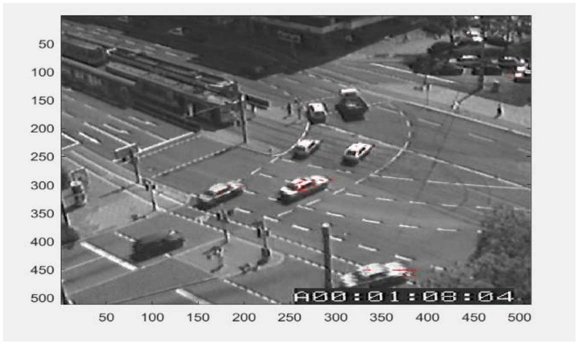
Optical flow or optic flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer (an eye or a camera) and the scene. To estimate the velocities spatial and temporal gradients should be used in x and y directions.

In our code we started by using a smoothing filter to reduce noise. We created our x and y filter kernels then we used these to convolute filtered image with x and y filters separately. We need to add same parameter to get the same size output otherwise it will compute full size of the matrix. After this we need to use our previous image to compute temporal gradient. If we do not put minus before Current-Previous the directions would be the opposite. Then we define x and y dimensions of the image. We create empty Vx and Vy velocity matrices same size as the image and empty G and b to use later. Then we compute G and b while looping the matrix.

$$G = \begin{bmatrix} \sum\limits_{p\in W} I_x^2 & \sum\limits_{p\in W} I_x I_y \\ \sum\limits_{p\in W} I_x I_y & \sum\limits_{p\in W} I_y^2 \end{bmatrix} \quad b = \begin{bmatrix} \sum\limits_{p\in W} I_x I_t \\ \sum\limits_{p\in W} I_y I_t \end{bmatrix}$$

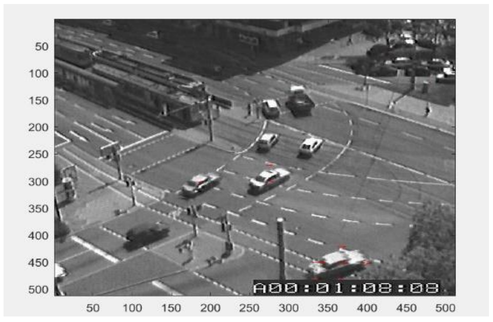
Then we need to compute u with velocity matrices. To compute u we need to multiply inverse of G and b but only if the minimum of eigen values of the G is bigger than the threshold then we can apply Vx and Vy. After all these steps we can compute our figure by using Vx and Vy.



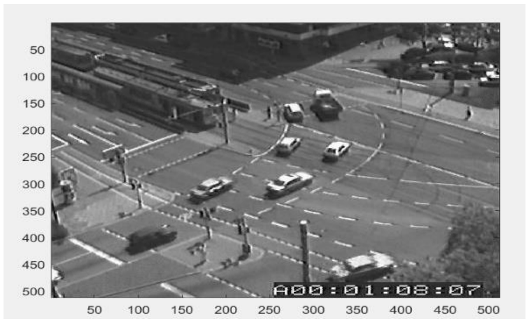


Box filter, k=30, t= 2*10^6 **Box filter, k=10, t= 2*10^6**

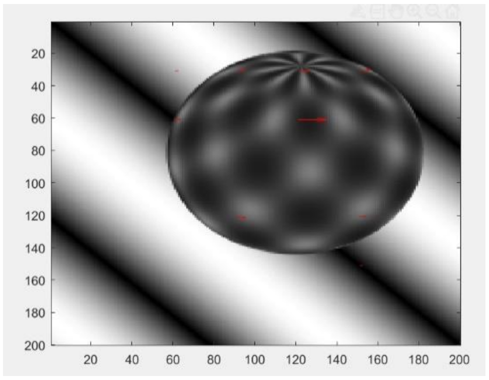
K is the parameter for sliding kernel size so when we make it bigger the number of pixels that are taken into consideration gets bigger. So the result gets affected by the neighbor pixels more because of this when we apply bigger k we can see that we are getting more arrow in the area.



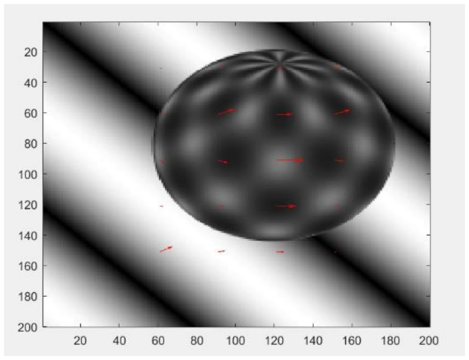
Gaussian filter, k=30, t= 2*10^6



Gaussian filter, k=10, t= 2*10^6



Box filter, k=30, t= 2*10^6



Box filter, k=30, t= 2*10^4

When we apply smaller threshold since there are more pixels that can exceed it, there will be more red arrows.

Appendix:

A.labOFMain.m

```
clear all; close all; clc;

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

Seq = sphere;

[row,col,num]=size(Seq);

% Define k and Threshold
k=30;  Threshold = 2*10^4;   % window size paramater


for j=2:1:num
ImPrev = Seq(:,:,j-1);  % first image
ImCurr = Seq(:,:,j);    % second image
lab6OF(ImPrev,ImCurr,k,Threshold);
pause(0.1);
end
```

B.labOFm.m

```
function lab6OF(ImPrev,ImCurr,k,Threshold)
    % Smooth the input images using a Box filter -builtin
    %Filtered_Curr = imgaussfilt(double(ImCurr),3);
    %Filtered_Prev = imgaussfilt(double(ImPrev),3);

    Filtered_Curr = imboxfilt(double(ImCurr),3);
    Filtered_Prev = imboxfilt(double(ImPrev),3);

    % Calculate spatial gradients (Ix, Iy) using Prewitt filter
- use
    % conv2(Im, filter) not builtin prewitt
    xFilter = [-1 0 1; -1 0 1; -1 0 1];
    yFilter = [-1 -1 -1; 0 0 0; 1 1 1];

    Ix = conv2(Filtered_Curr, xFilter, 'same');
    Iy = conv2(Filtered_Curr, yFilter, 'same');

    %Spatial --> current frame
    %Temporal --> using both current and prev
    %temporal = that evolving with time not with place - one
line code

    % Calculate temporal (It) gradient - use prev image here
only
    It = -(Filtered_Curr - Filtered_Prev);

    [yDim,xDim] = size(Filtered_Curr);
    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
            subIx = Ix(y-k:y+k,x-k:x+k);
            subIy = Iy(y-k:y+k,x-k:x+k);
            subIt = It(y-k:y+k,x-k:x+k);

            % Calculate the elements of G and b
            G = [sum(sum(subIx.^2)) sum(sum(subIx.*subIy));
sum(sum(subIx.*subIy)) sum(sum(subIy.^2))];
            b = [sum(sum(subIx.*subIt));
sum(sum(subIy.*subIt))];
```

```
            if (min(eig(G)) > Threshold)
                % Calculate u
                u = -(inv(G)).*b;
                Vx(y,x)=u(1);
                Vy(y,x)=u(2);
            end
        end
    end

    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; %add drawnow when you comment out
pause
end
```