OPTICAL FLOW

DECEMBER 7, 2021

TA's: Muhammed Zemzemoğlu and Mehmet Emin Mumcuoğlu

Name of Student: Moses Chuka Ebere
Student Number: 31491
Course: Computer Vision (EE 417) Lab

TABLE OF CONTENTS

INTRODUCTION

Optical flow, in the most basic terms, is the **apparent motion of brightness patterns** observed in an image sequence. In the most ideal case, it would be nice to estimate motion field which is the projection of relative 3D motion of an object onto the image plane. However, it has been found to be overly difficult to calculate this; hence, it is common to assume that the apparent brightness of moving objects remains constant. By extension, we calculate optical flow instead.

Optical flow has found relevance in areas like object detection and tracking, image dominant plane extraction, movement detection, robot navigation and visual odometry.

Today's lab focuses on calculating optical flow throughout a number of videos, while measuring the velocities of objects in the videos.

EXPLANATION OF METHODS

OPTICAL FLOW

To calculate the apparent motion of brightness patterns, local deformation models like translational model, affine model, transformation of intensity values and occlusions, etc., could be used.

For the purpose of this lab, a simple translational model was used. The model is governed by the following equation:

$$I_1(\mathbf{x}_1) = I_2(\mathbf{x}_1 + \Delta\mathbf{x})$$

We now index our video sequence with time to give:

$$I(\mathbf{x}(t), t) = I(\mathbf{x}(t) + \mathbf{u}dt, t + dt)$$

Notice: We assume constant intensity values across consecutive frames.
Through Taylor series expansion, we obtain the **brightness constancy constraint** (BCC):

$$\nabla I(\mathbf{x}(t), t)^T \mathbf{u} + I_t(\mathbf{x}(t), t) = 0$$

Since the above is an ill-defined problem given that there are two unknowns (since u is the velocity vector for the pixel in question) in one equation, we can approach this as an optimization problem.

Kanade-Tomasi suggested that a window be considered (with uniform-intensity pixels to ensure constant OF). Within the window, the sum of squared errors is minimized.

$$E_b(\mathbf{u}) = \sum_{W(x,y)} [\nabla I^T(x,y,t)\mathbf{u}(x,y) + I_t(x,y,t)]^2$$

This yields a unique result:

$$\mathrm{Gu} + \mathrm{b} = 0$$

Where:

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

$$u = [Vx; Vy] = -G^{-1}b$$

In essence, the following are performed on two concurrent image frames:

1. Smooth the input images.
2. Calculate their gradients – spatial gradient and temporal gradient.
3. Within a window, compute G and b.
4. Using the above results, compute the optical flow of the pixel under consideration.

Prior to the above, the individual frames of the video must be obtained, and two concurrent frames are used for the optical flow algorithms.

In MATLAB, two scripts were written to perform the above tasks. The MainScript functions to separate video frames, while the function is used to calculate the optical flow. The two scripts are included in the appendix.

IN-LAB RESULTS

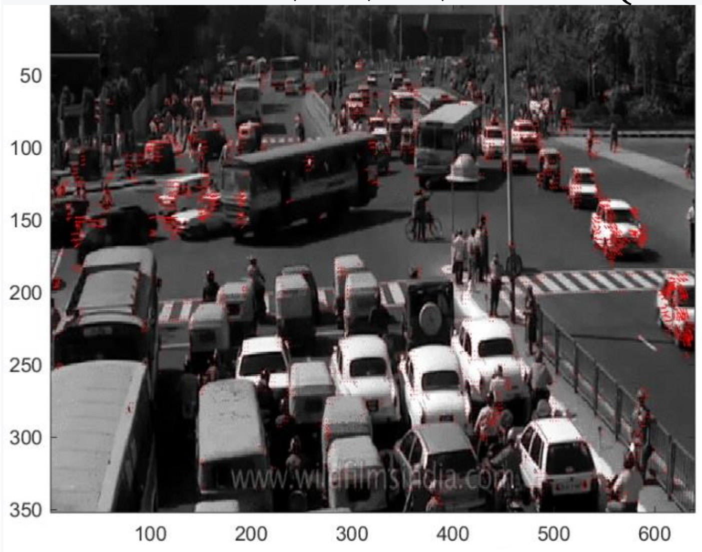
Optical Flow

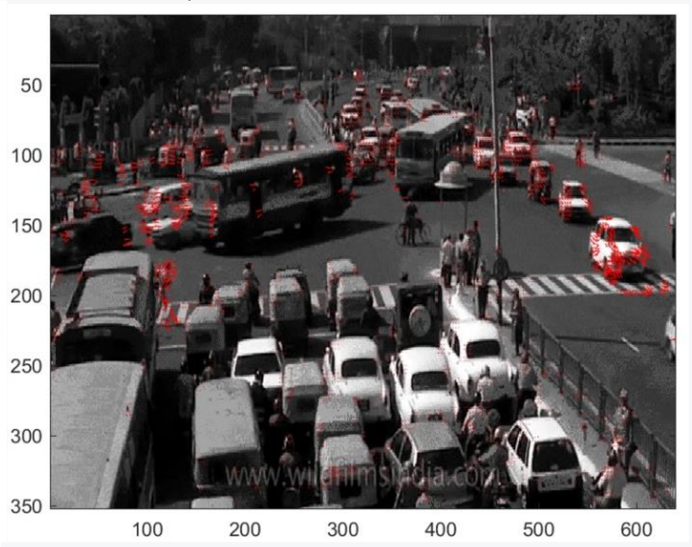
The explanation of how I obtained the resulting images for each frame is contained in the Post-lab questions sections.

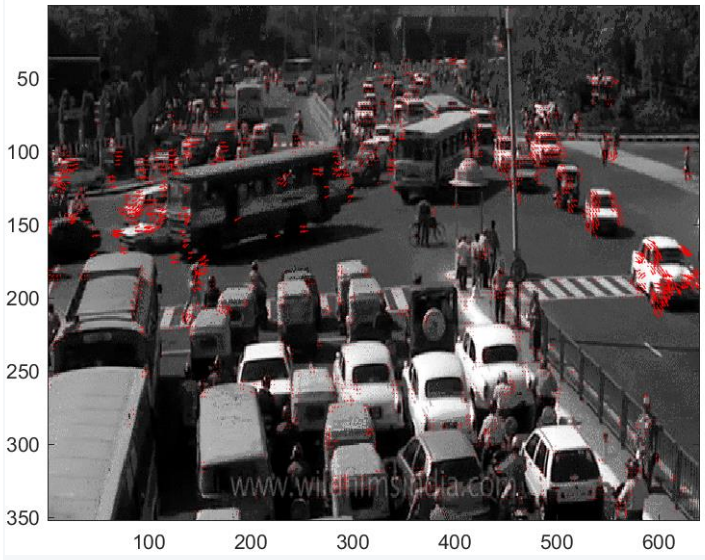
For this section, I picked four images obtained for each data set.

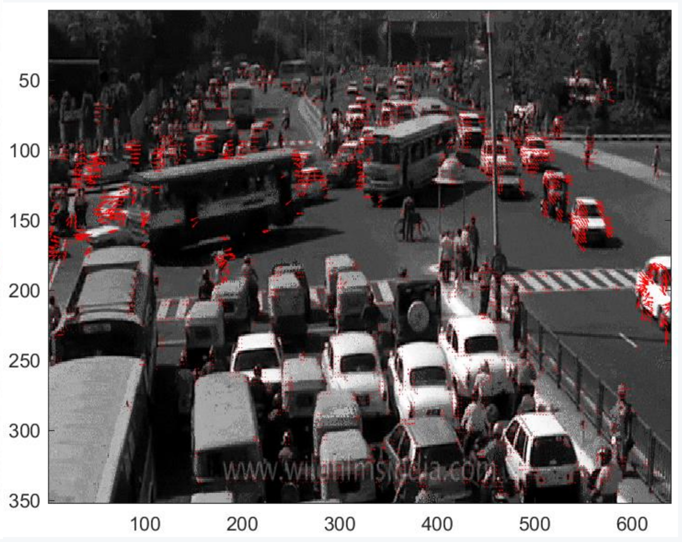
Cars1

There are 33 frames in the cars1 data set, so I've attached the images from frames 2, 10, 20, and 30 (in that order) below:



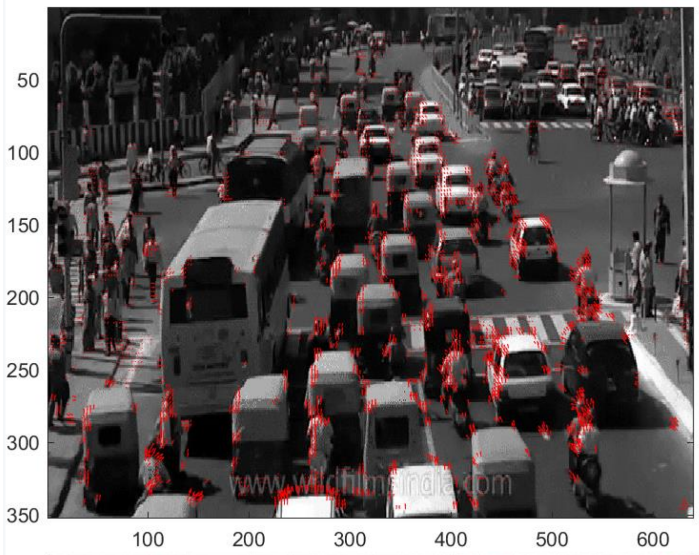
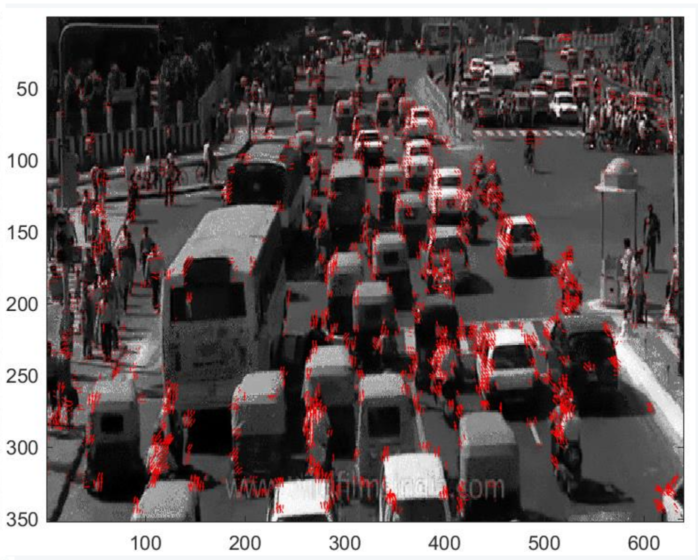
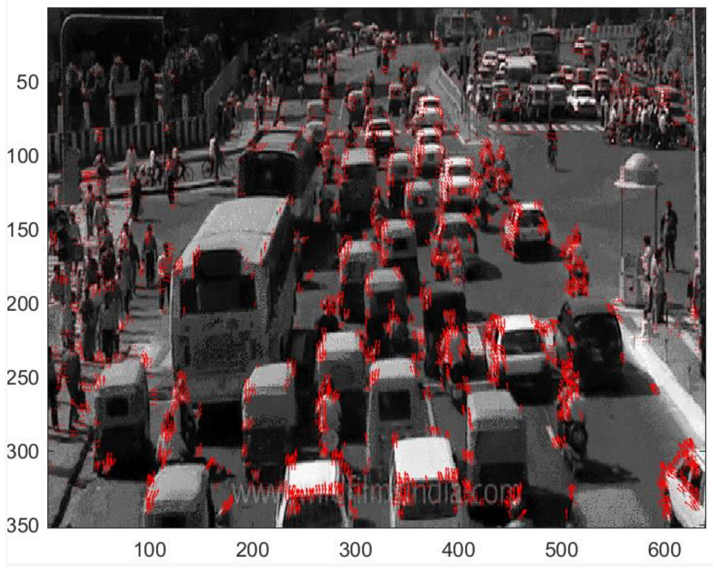
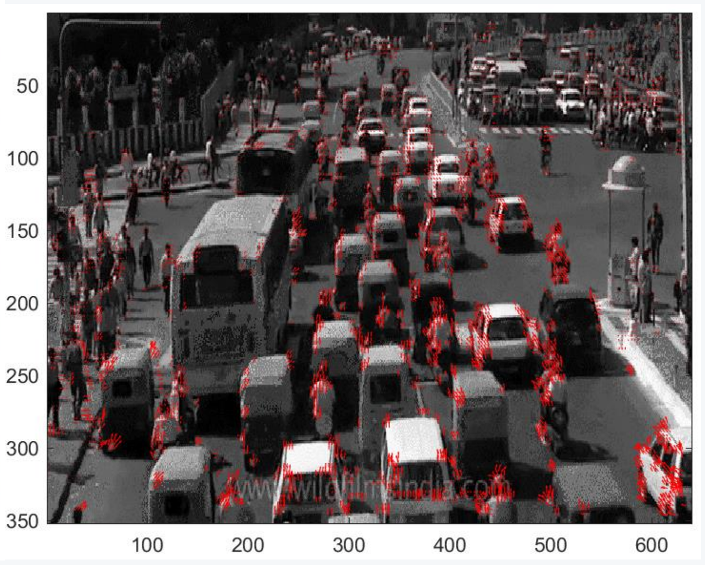






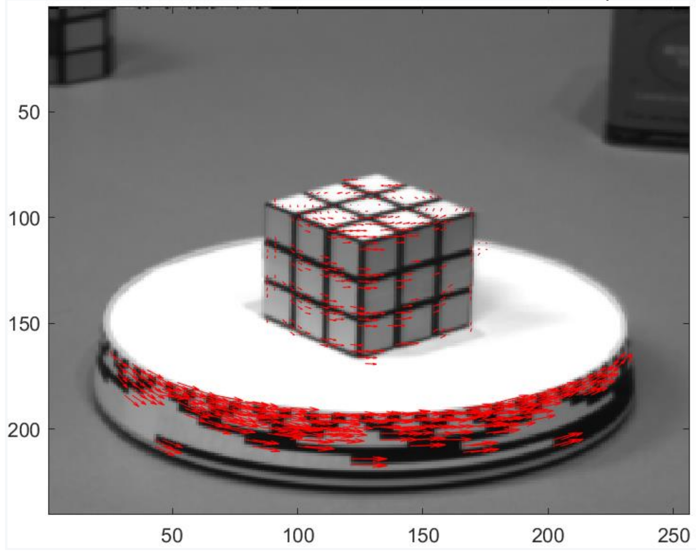
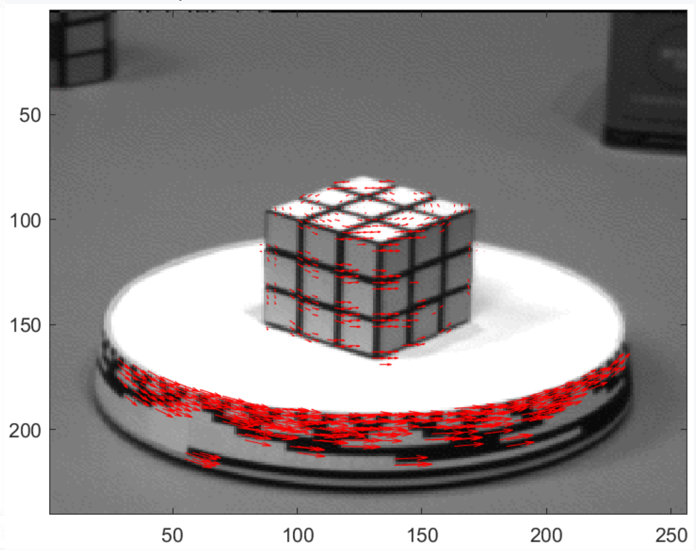
Cars2

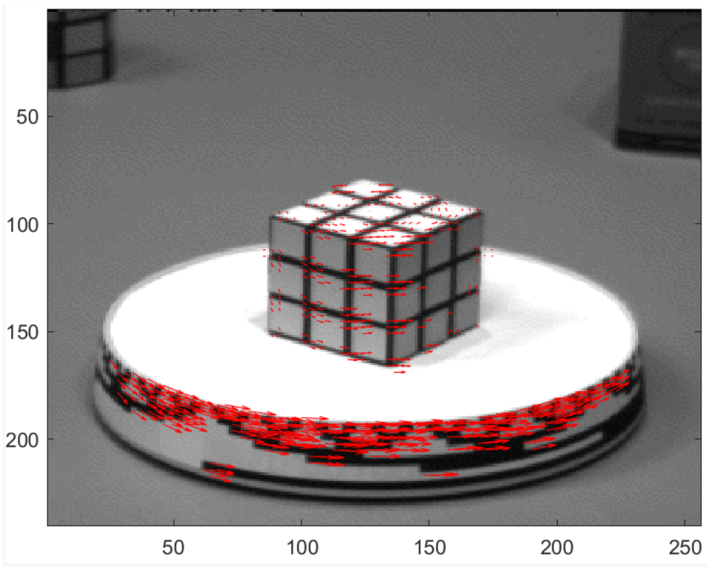
There are 25 frames in the cars2 data set, so I've attached the images from frames 1, 8, 17, and 24 (in that order) below:

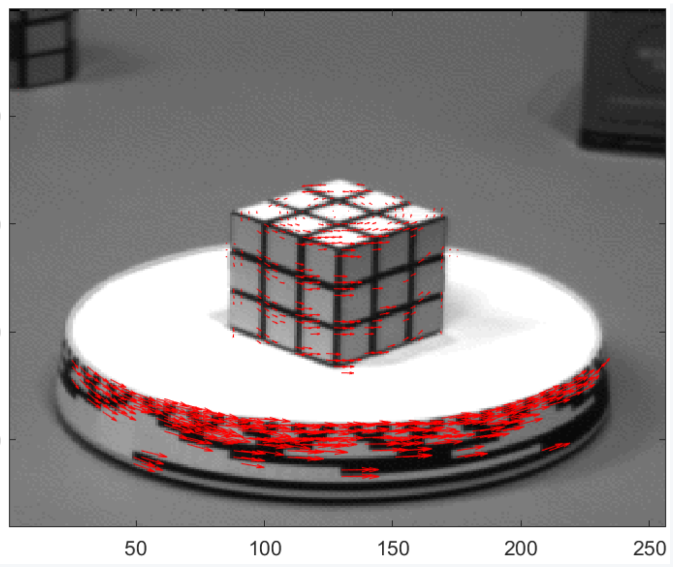





Rubic

There are 19 frames in the rubic data set, so I've attached the images from frames 1, 6, 11, and 17 (in that order) below:

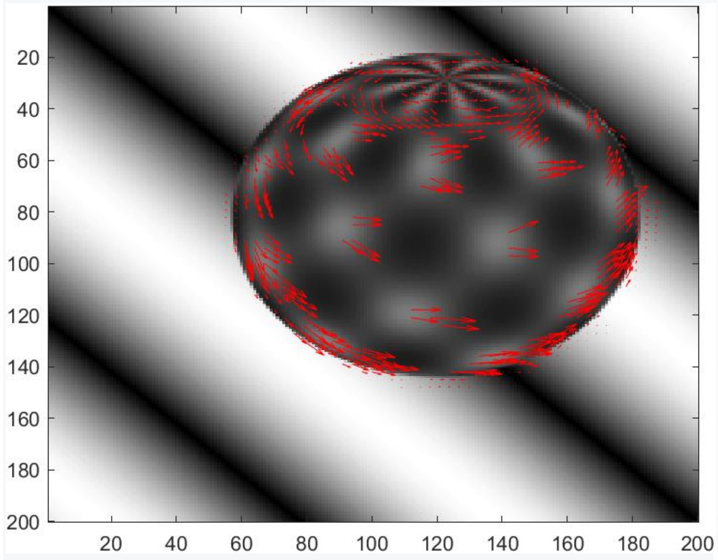



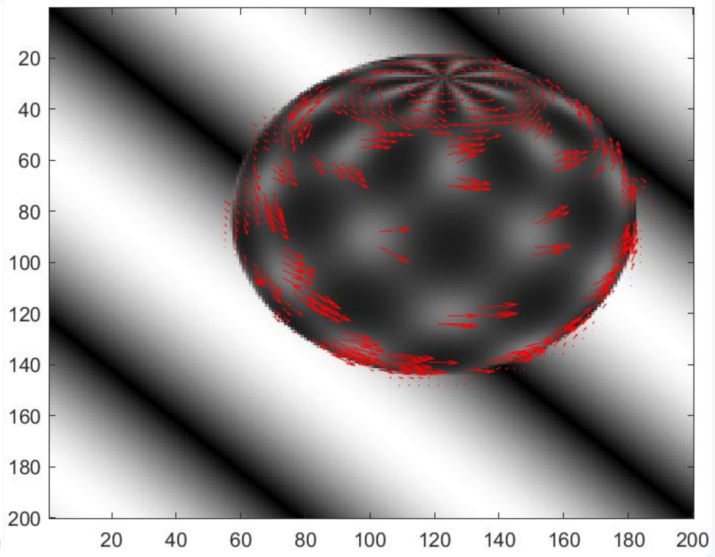


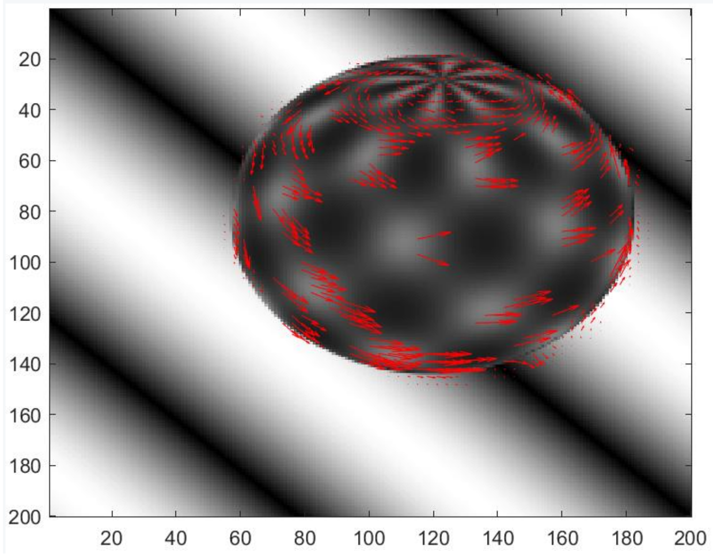


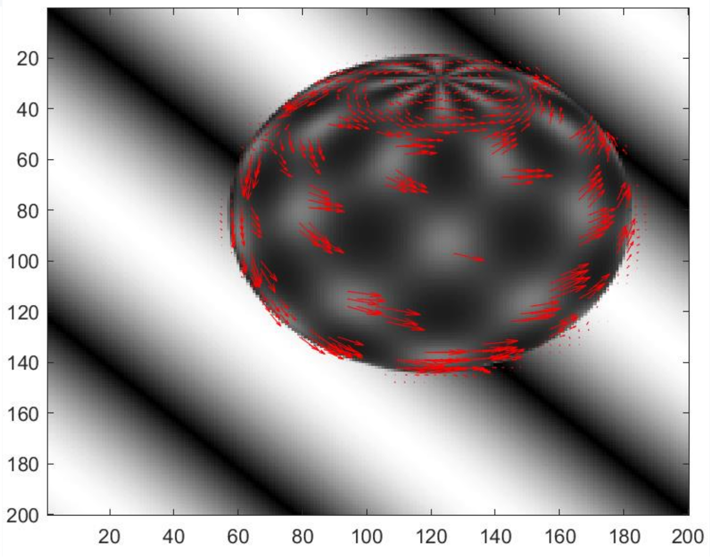
Sphere

There are 18 frames in the sphere data set, so I've attached the images from frames 1, 6, 11, and 17 (in that order) below:



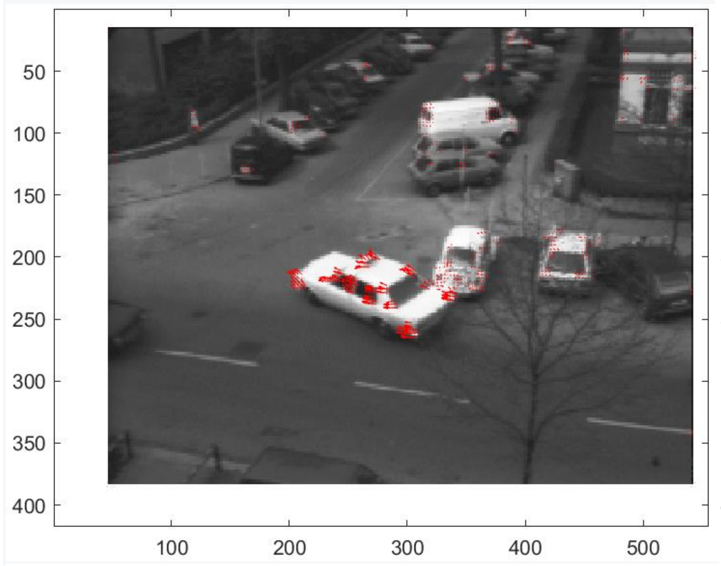


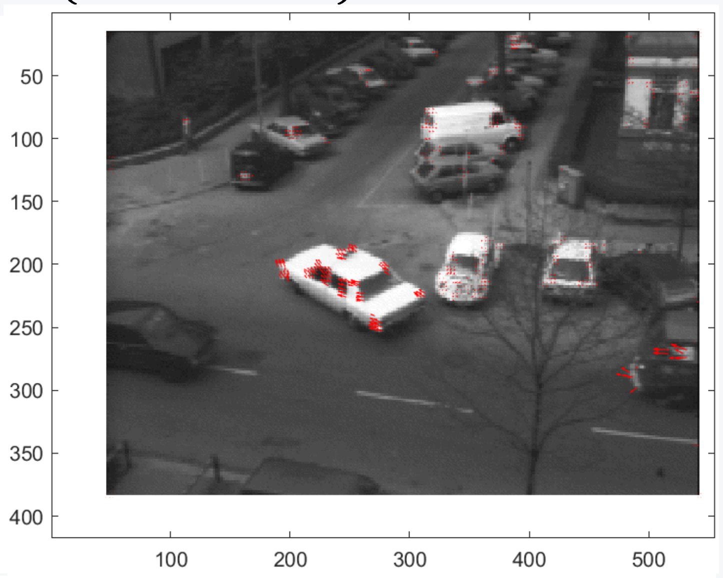


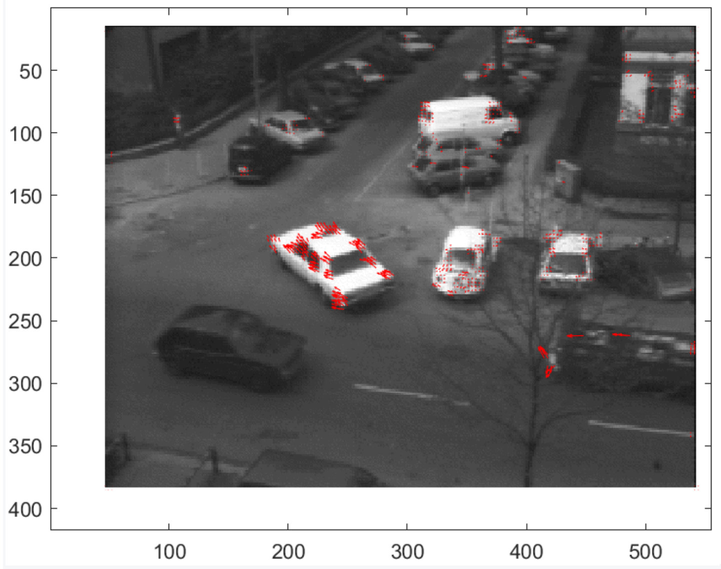


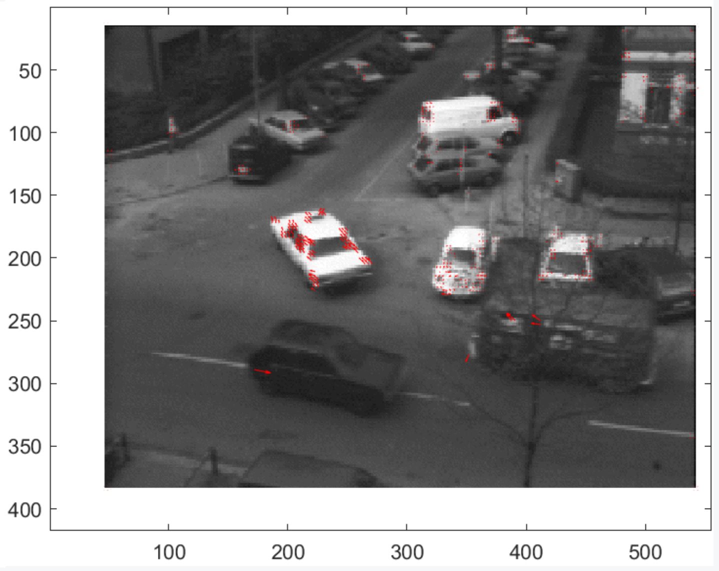
Taxi

There are 40 frames in the sphere data set, so I’ve attached the images from frames 1, 13, 26, and 39 (in that order) below:



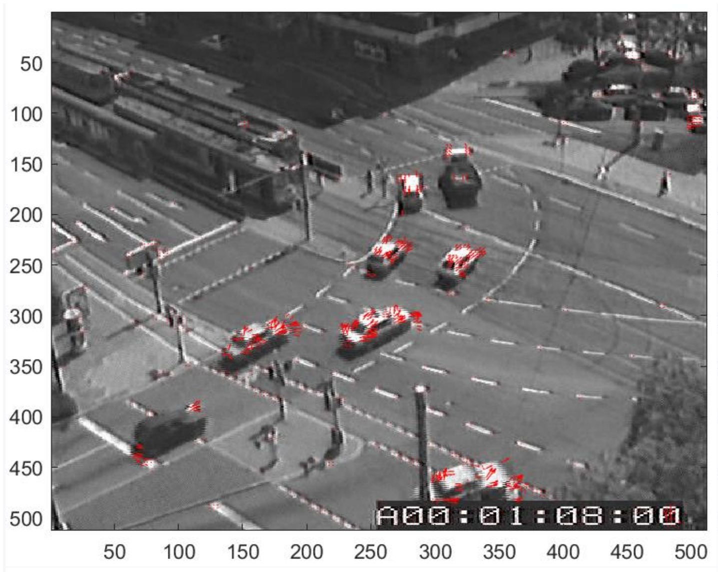
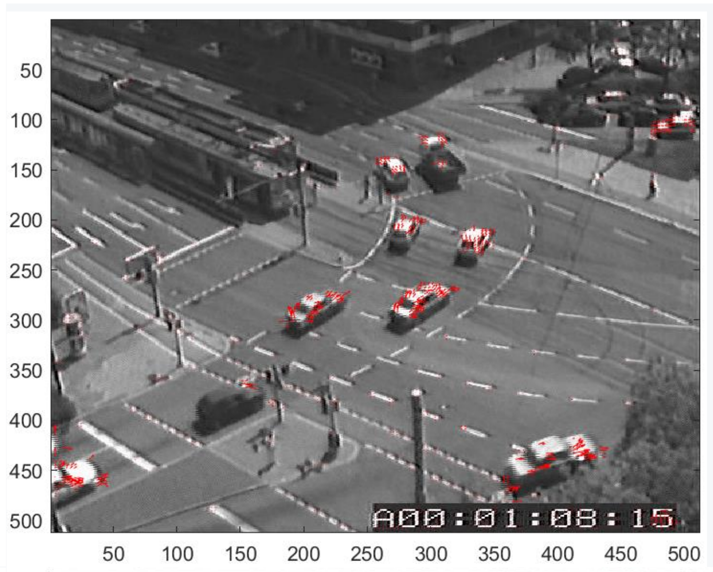
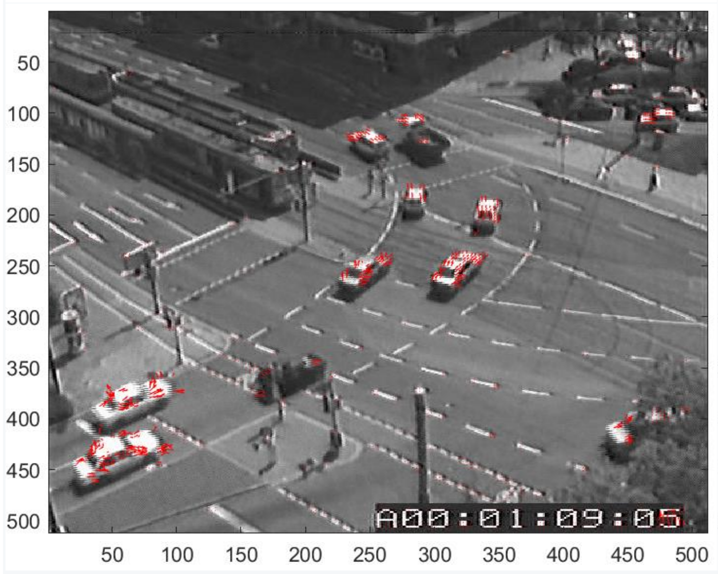
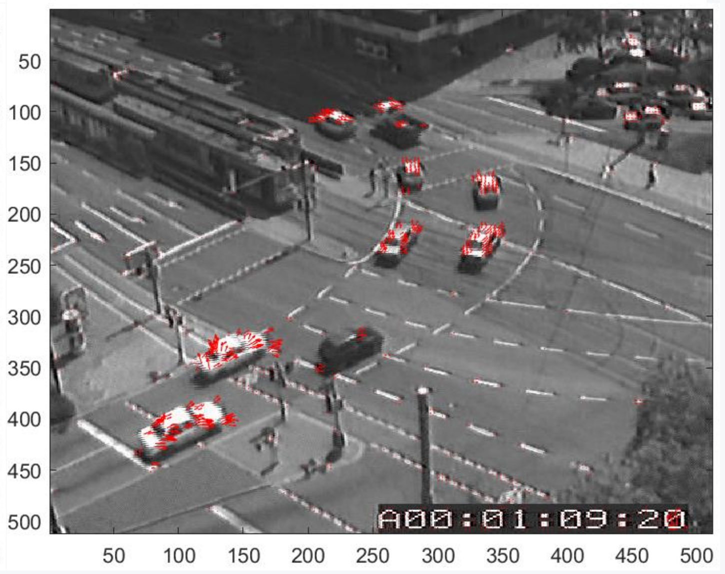






Traffic

There are 47 frames in the sphere data set, so I’ve attached the images from frames 1, 16, 31, and 46 (in that order) below:


A00:01:08:00

A00:01:08:15

A00:01:09:05

A00:01:09:20

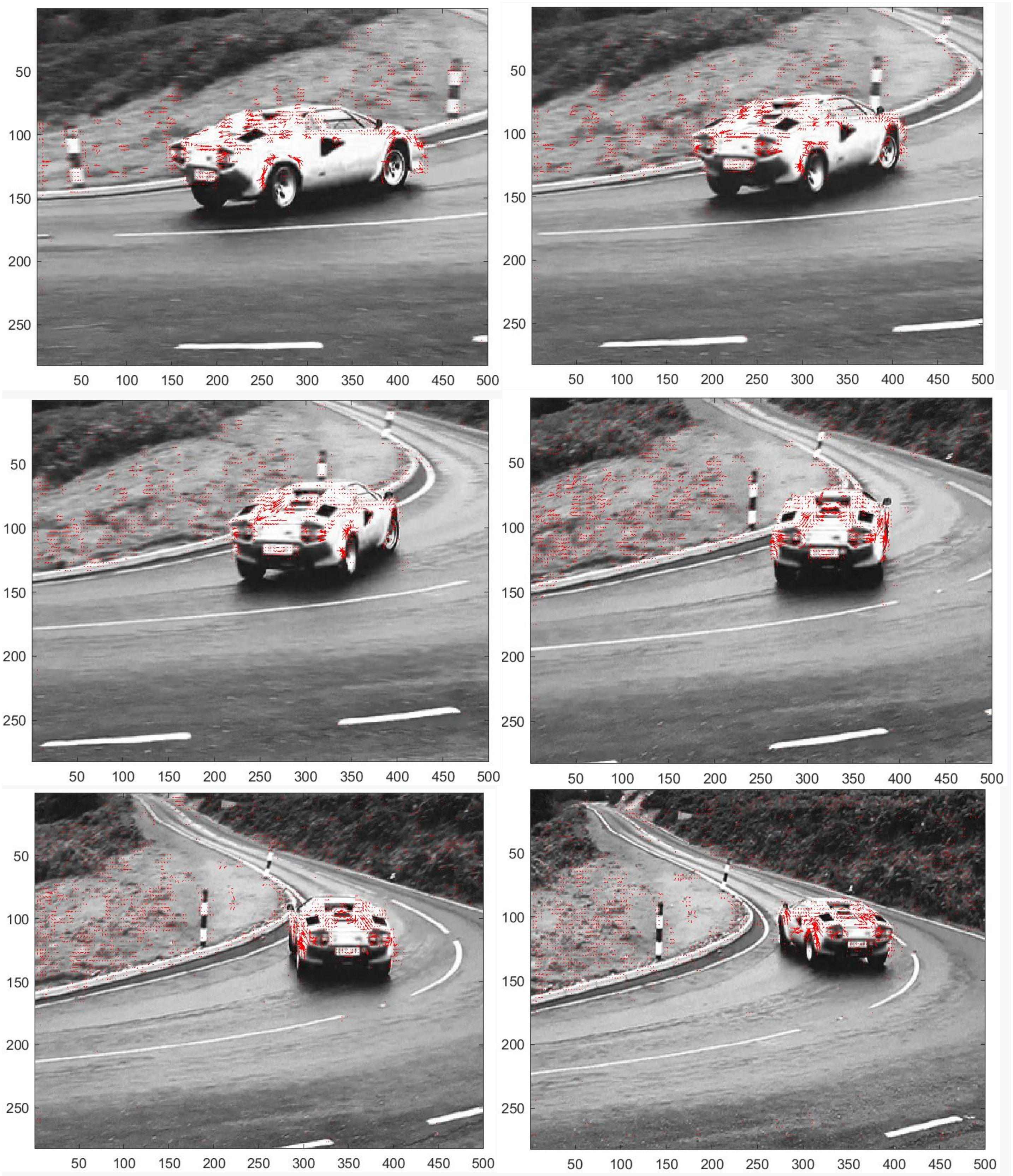
MULTIPLE RESULTS

I downloaded a gif image of drifting car, converted it to a video and tested the optical flow algorithm on it.

I had difficulties converting the video file to .mat format for the algorithm, so I simply read the video frame-by-frame and then perform the optical flow calculation. The code used is the following:

```
clear all; close all; clc;

% read in the video
v = VideoReader('e.mp4');
% Extract some frames for the optical flow calculation.
frames = read(v,[5 20]);

Seq = frames;

% The 'num' variable corresponds to the frame number.
[row,col,num]=size(Seq);

% Define k (window size) and Threshold
k = 3;
Threshold = 9000;

for j=2:1:num
    ImPrev = Seq(:,:,j-1);
    ImCurr = Seq(:,:,j);
    h = lab7OF_Post_lab(ImPrev, ImCurr, k, Threshold);
    % To save every frame.
    saveas(h, sprintf('FIG%d.png',j));
    pause(0.5);
end
```

These are six frames obtained:



POST-LAB QUESTIONS

Procedure Followed to Obtain the Resulting Images for Each Step

- Firstly, within the optical flow function, I created a variable containing the image with scaled colors.

```
h = imagesc(ImPrev);
```

- I did this to be able to retain the resulting image at every instance.
- Now, I altered the function definition to return the image in h using:

```
output = h;
```

  where the output variable is returned whenever the function is called.
- In the main script, I obtained the output image while calling the function, and using the **saveas function**, I saved every frame produced:

```
 h = lab7OF_Post_lab(ImPrev, ImCurr, k, Threshold);
                % To save every frame.
        saveas(h, sprintf('FIG%d.png',j));
```

- Notice: `sprintf('FIG%d.png',j)`
- In included the above syntax to save each frame separately.

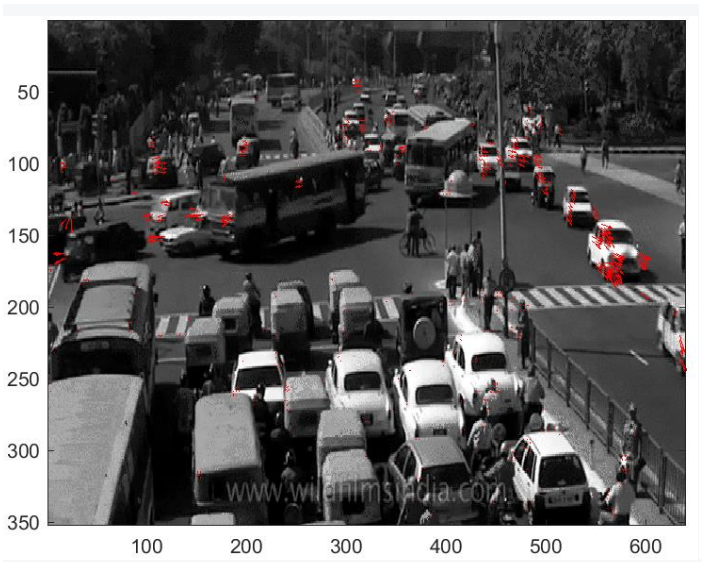
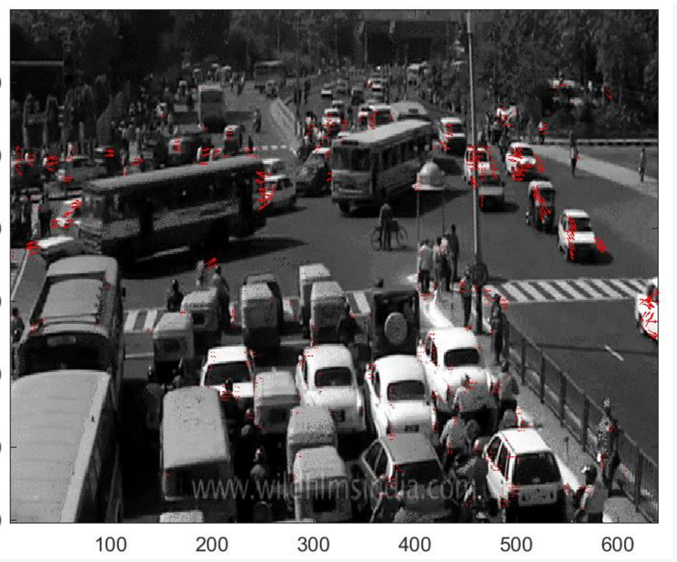
included For subsequent post-lab questions, I used an if statement to just obtained a limited amount of images for comparison.

```
if mod(j,6) == 0
        saveas(h, sprintf('FIG%d.png',j));
    end
```

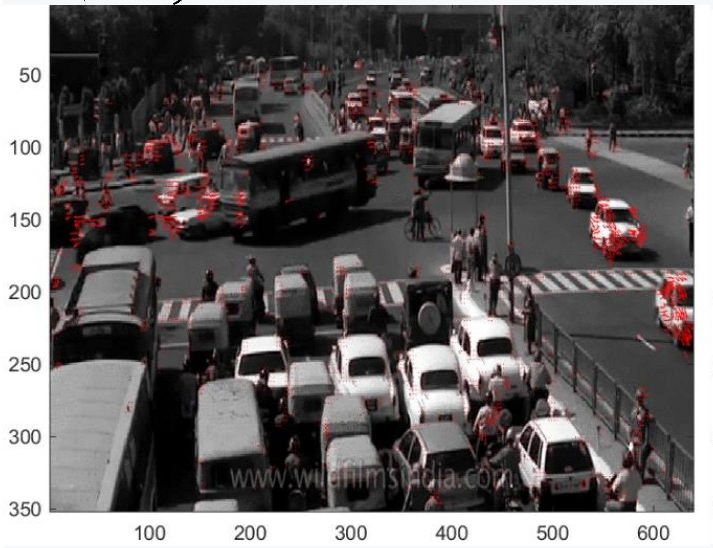
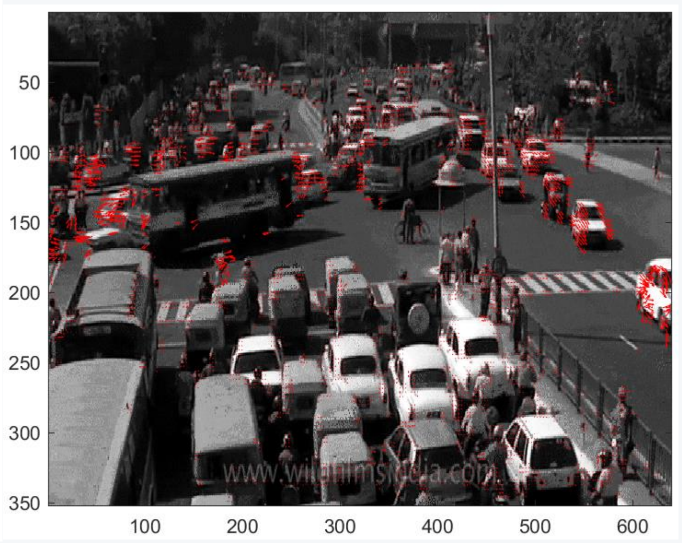
Smoothing Filters

A Gaussian filter was used, and the results were compared with those obtained using a box filter.

Cars1

The above were obtained with a Gaussian filter (threshold = 200,000).

The above were obtained with a box filter (threshold = 5,000,000).

Observations:

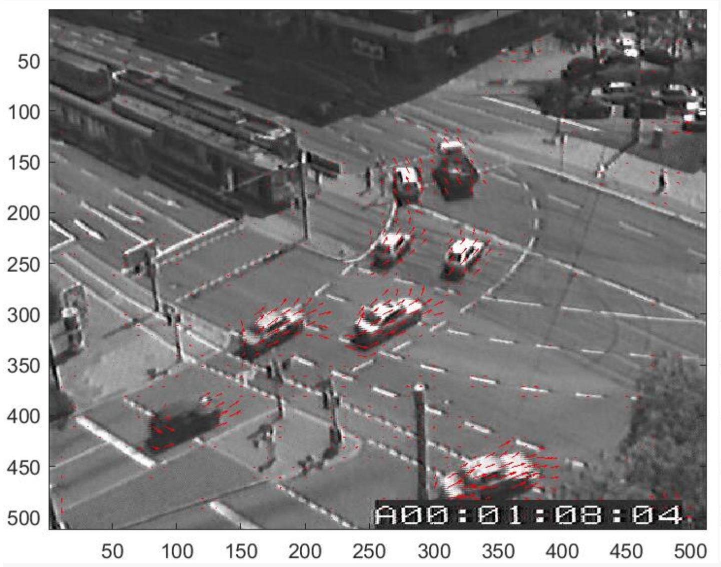
- Obviously, the Gaussian filter does a better job at smoothing than the box filter.
- A **significant observation is that for the same window size, the ideal threshold needs to be altered when the smoothing filter is changed.** ***Notably, a lower threshold is needed when the Gaussian filter is used.***
- The results appear better with the Gaussian filter.

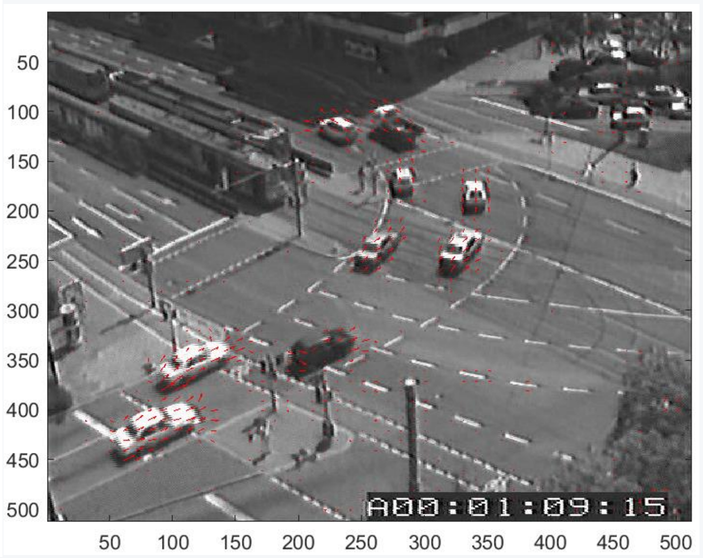
Window Sizes

Different windows sizes (k = 10, 20, and 30) were applied to see the effect on the results.

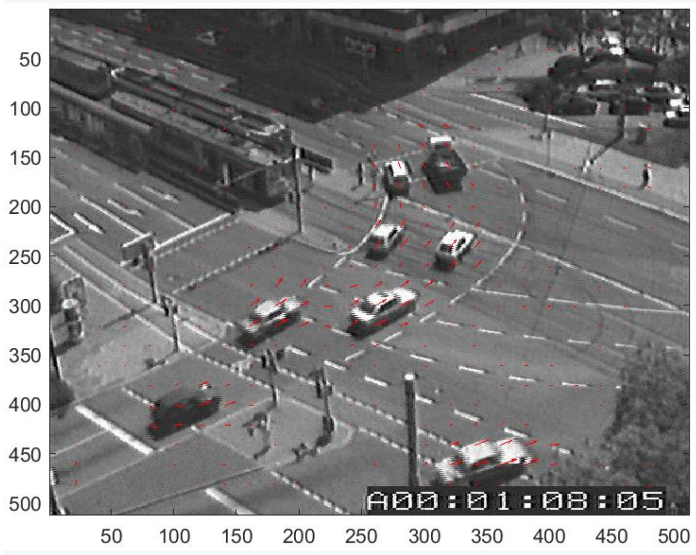
I applied this on the traffic dataset to obtain the following:

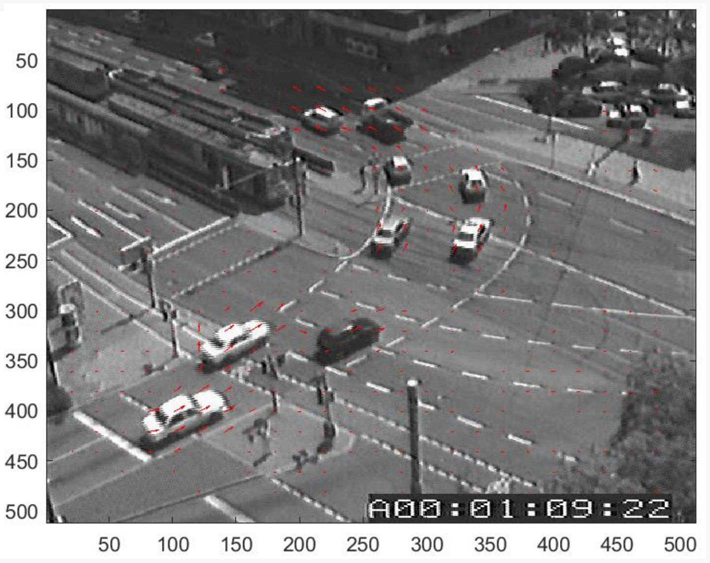
For k = 10



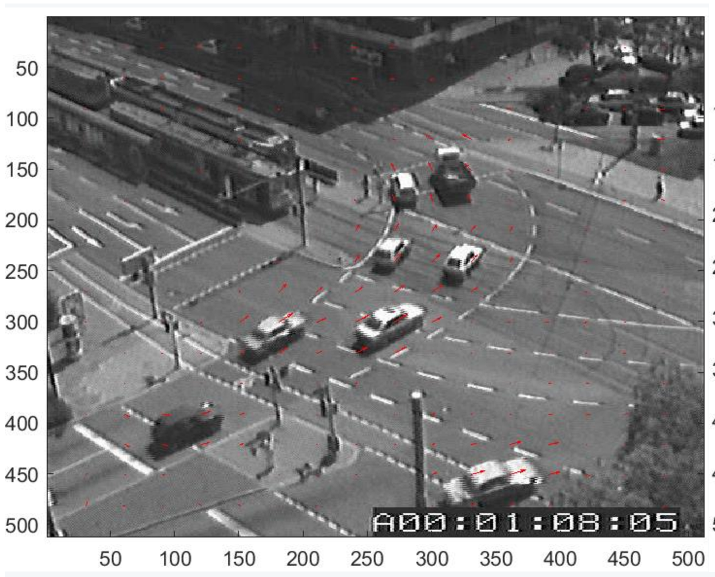


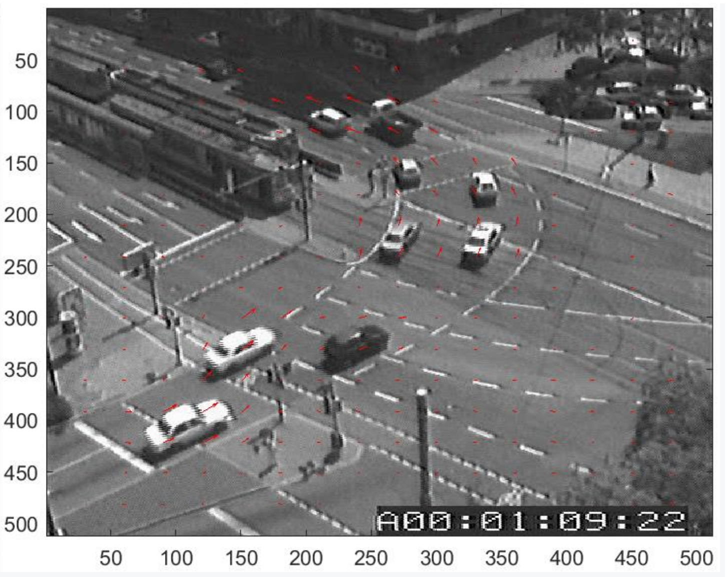
For k = 20





For k = 30





Observations:

- As the window size increases, the spacing of the detections also increases. As seen in the above results, the arrows become farther apart.
- I would say that the accuracy of the optical flow algorithm reduces as we increase the window size.
- Why? The notion of optical flow is to observe apparent motion. While k is increased, some optical flow vectors are detected on original still parts of the video (like on the road surface). The road is not moving, so there should be no vector on it. The vectors are supposed to be attached to apparently moving items.
- This was also due to the major assumption that all pixels within a window have the same intensities and optical flow vectors.

DISCUSSION

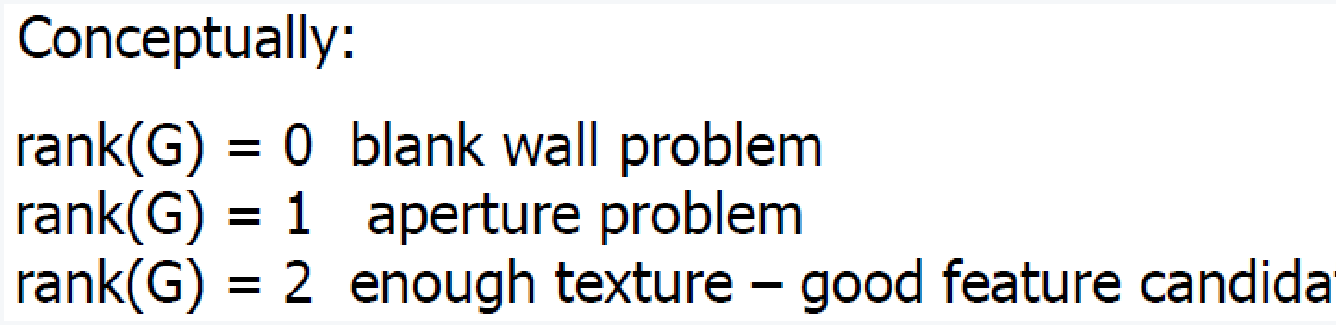
- The filter used for the optical flow algorithm directly affects some other parameters like the required threshold. As mentioned in the previous section, lower threshold values are needed when a Gaussian filter is used.
- The window size also sort of determines the accuracy of the results.

- Optical flow is clearly not motion field. Since motion field is overly difficult to estimate, we calculate optical flow instead.
- For this particular lab, the local deformation model used was a translational model; however, for a more robust algorithm, it might be better to use models like the affine model (which account for other possibilities like rotation, scaling, shearing, etc.) or occlusion-incorporating models which account for occlusion and intensity value changes.

- Optical flow, although not accurate, is good for feature tracking.

- A major assumption was that all pixels within a window have the same intensities and optical flow vectors.

- Our G matrix is particularly important because it determines what we can actually calculate (by checking its rank).


Conceptually:

rank(G) = 0 blank wall problem
rank(G) = 1   aperture problem
rank(G) = 2 enough texture – good feature candida

REFERENCES

Reinhard Klette. 2014. Concise Computer Vision: An Introduction into Theory and Algorithms. Springer Publishing Company, Incorporated.

https://en.wikipedia.org/wiki/Optical_flow

https://www.mathworks.com/help/matlab/ref/imagesc.html

https://ezgif.com/gif-to-mp4/ezgif-6-2f6c62437c20.gif

https://giphy.com/explore/stock

https://ch.mathworks.com/help/matlab/ref/videoreader.html

APPENDIX

OPTICAL FLOW

Two m-files were created for this lab:

In-Lab Code

Main Script

```
clear all; close all; clc;

% Load the files
load('cars1.mat');
load('cars2.mat');
load('rubic.mat');
load('sphere.mat');
load('taxi.mat');
load('traffic.mat');

Seq = taxi;

% The 'num' variable corresponds to the frame number.
[row,col,num]=size(Seq);

% Define k (window size) and Threshold
k = 3;
Threshold = 5000000;

for j=2:1:num
    ImPrev = Seq(:,:,j-1);
    ImCurr = Seq(:,:,j);
    lab7OF(ImPrev, ImCurr, k, Threshold);
    pause(0.1);
end
```

Function

```
function lab7OF(ImPrev, ImCurr, k, Threshold)
% Smooth the input images using a Box filter
Box_filt = [1 1 1; 1 1 1; 1 1 1];
ImPrev1 = conv2(ImPrev, Box_filt);
ImCurr1 = conv2(ImCurr, Box_filt);
```

```
% Calculate spatial gradients (Ix, Iy) using Prewitt filter
x_filt = [1 0 -1; 1 0 -1; 1 0 -1];
y_filt = [1 1 1; 0 0 0; -1 -1 -1];
Ix = conv2(ImCurr1, x_filt,'same');
Iy = conv2(ImCurr1, y_filt,'same');

% Calculate temporal (It) gradient
It = ImCurr1 - ImPrev1;

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

for x=k+1:k:xdim-k-1
    cy=k+1;
    for y=k+1:k:ydim-k-1
        % Create a window of the image gradients (from -k
to +k)
        wpx = Ix(y-k:y+k, x-k:x+k);
        wpy = Iy(y-k:y+k, x-k:x+k);
        % Create the corner matrix, G.
        G(1,1) = sum(sum(wpx.*wpx));
        G(1,2) = sum(sum(wpx.*wpy));
        G(2,1) = G(1,2);
        G(2,2) = sum(sum(wpy.*wpy));
        % Find the eigenvalues of G.
        L1 = eig(G);

        wpt = It(y-k:y+k, x-k:x+k);
        b(1,1) = sum(sum(wpt.*wpx));
        b(2,1) = sum(sum(wpt.*wpy));

        if (min(L1) < Threshold)
            Vx(cy,cx)=0;
            Vy(cy,cx)=0;
        else
            % Calculate u
            u = -1*(G\b);
            Vx(cy,cx)=u(1);
```

```
                Vy(cy,cx)=u(2);
            end
                cy=cy+k;
        end
        cx=cx+k;
    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;
    end
```

Post-Lab Code

Main Script

```
clear all; close all; clc;

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

Seq = cars1;

% The 'num' variable corresponds to the frame number.
[row,col,num]=size(Seq);

% Define k (window size) and Threshold
k = 3;
Threshold = 90000;

for j=2:1:num
    ImPrev = Seq(:,:,j-1);
    ImCurr = Seq(:,:,j);
    h = lab7OF_Post_lab(ImPrev, ImCurr, k, Threshold);
    % To save every frame.
    saveas(h, sprintf('FIG%d.png',j));
    pause(0.1);
end
```

Function

```
function [output] = lab7OF_Post_lab(ImPrev, ImCurr, k,
Threshold)
% Smooth the input images using a Box filter
Box_filt = [1 1 1; 1 1 1; 1 1 1];
% ImPrev1 = conv2(ImPrev, Box_filt);
% ImCurr1 = conv2(ImCurr, Box_filt);

%create the gaussian filter
Gauss_filt = [1 4 7 4 1; 4 16 26 16 4; 7 26 41 26 7; 4 16
26 16 4; 1 4 7 4 1]/273;
ImPrev1 = conv2(ImPrev, Gauss_filt);
ImCurr1 = conv2(ImCurr, Gauss_filt);

% Calculate spatial gradients (Ix, Iy) using Prewitt filter
x_filt = [1 0 -1; 1 0 -1; 1 0 -1];
y_filt = [1 1 1; 0 0 0; -1 -1 -1];
Ix = conv2(ImCurr1, x_filt,'same');
Iy = conv2(ImCurr1, y_filt,'same');

% Calculate temporal (It) gradient
It = ImCurr1 - ImPrev1;

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

for x=k+1:k:xdim-k-1
    cy=k+1;
    for y=k+1:k:ydim-k-1
        % Create a window of the image gradients (from -k
to +k)
        wpx = Ix(y-k:y+k, x-k:x+k);
        wpy = Iy(y-k:y+k, x-k:x+k);
        % Create the corner matrix, G.
        G(1,1) = sum(sum(wpx.*wpx));
        G(1,2) = sum(sum(wpx.*wpy));
        G(2,1) = G(1,2);
        G(2,2) = sum(sum(wpy.*wpy));
        % Find the eigenvalues of G.
        L1 = eig(G);

        wpt = It(y-k:y+k, x-k:x+k);
        b(1,1) = sum(sum(wpt.*wpx));
        b(2,1) = sum(sum(wpt.*wpy));

        if (min(L1) < Threshold)
            Vx(cy,cx)=0;
            Vy(cy,cx)=0;
        else
            % Calculate u
            u = -1*(G\b);
            Vx(cy,cx)=u(1);
            Vy(cy,cx)=u(2);
        end
            cy=cy+k;
    end
    cx=cx+k;
end

cla reset;
h = imagesc(ImPrev); hold on;
[xramp,yramp] = meshgrid(1:1:xdim,1:1:ydim);
quiver(xramp,yramp,Vx,Vy,10,'r');
colormap gray;

output = h;
end
```