Computational Methods for Data Analysis

Final Project

Due on March 19, 2020 at 5:00pm

Professor J. Nathan Kutz

Section A

Chandan Sharma Subedi

Abstract

In this paper, Singular Value Decomposition is applied to stabilize a video, captured with a hand-held camera, while walking. Optical flow is estimated between two consecutive frames to construct the matrix. Left singular matrix is used to identify the mode associated with pixels moving up and down due to the act of walking. The video is then reconstructed using first few principal components. Lucas-Kanade and Horn-Schunck methods are experimented using MATLAB built-in functions for better optical flow estimates.

1 Introduction and Overview

Everyday Terabytes of videos are captured all over the world across many devices. This has attracted the development of many video stabilization softwares. The iPhone has it's own video stabilization software that runs in real-time as you take the video. In this paper, an attempt has been made to investigate the application of SVD to stabilize a particular type of instability, distortions on the video due to walking motion. When a person walking on the road captures a video through a hand-held camera, you see images shifting up and down on the video. Is there a way we can identify that motion and remove it from the video?

A concept of Optical Flow is introduced that identifies how each pixels are moving between two consecutive frames. Using this velocity data, two matrices are constructed, each corresponding to x-direction and y-direction. Singular Value Decomposition is used to extract information from these matrices. Section 2 lays down theoretical foundation behind Optical Flow and SVD, while section 3 describes algorithms used for the analysis. Section 4 put forwards the result of analysis.

2 Theoretical Background

Optical flow is the motion of pixels between consecutive image frames, caused by the relative movement between the scene and the camera. It measures the velocity of each pixel in the image. Let $I(x, y, t)$ be the intensity of an image as a function of space (x, y) and time (t). If image is dispaced by $(\delta x, \delta y)$ over time t, the new image obtained is $I(x + \delta x, y + \delta y, t + \delta t)$. It is assumed that the intensities of the object do not change between consecutive frames. Thus,

$$I(x, y) = I(x + dx, y + dy, t + dt)$$

The Taylor series expansion of RHS about the point (x, y, t) gives us,

$$I(x, y, t) = I(x, y, t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t + ...$$
$$\implies \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t = 0$$

Dividing by dt, we get the optical flow equation:

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0$$

This equation cannot be solved directly since we have two unknowns (u, v), that specify the velocity of a pixel, but one equation. Lucas-Kanade and Horn-Schunck method are two techniques to compute sparse optical flow. Under Lucas-Kande method a $3x3$ patch of pixels are considered, instead of one, and it is assumed that all 9 pixels in the patch have the same velocities. This gives us an over-defined system of 9 linear equations that can be solved using least-square method.

$$Ax = b$$
$$\begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_9) & I_y(q_9) \end{bmatrix} \begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_9) \end{bmatrix}$$

Singular Value Decomposition is a matrix decomposition of the form

$$\begin{bmatrix} \\ \mathbf{A} \\ \\ \end{bmatrix}\begin{bmatrix} \\ \mathbf{v_1} & \mathbf{v_2} & \dots & \mathbf{v_n} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \mathbf{u_1} & \mathbf{u_2} & \dots & \mathbf{u_n} \\ \\ \end{bmatrix}\begin{bmatrix} \sigma_\mathbf{1} & & & \\ & \sigma_\mathbf{2} & & \\ & & \ddots & \\ & & & \sigma_\mathbf{n} \end{bmatrix}$$

$$\mathbf{AV} = \mathbf{U\Sigma}$$

$$\mathbf{A} = \mathbf{U\Sigma V}^*$$

The matrix $\mathbf{\Sigma}$ is an m x n diagonal matrix with positive entries provided the matrix $\mathbf{A}$ is of full rank. It stores the singular values, representing the stretching space, ordered from largest to smallest so that $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n$. The matrix $\mathbf{U}$ is a m x n unitary matrix with orthonormal columns (left-singular vectors). The matrix $\mathbf{V}$ is an n x n unitary matrix with orthonormal columns (right-singular vectors). Both of these vectors represent rotation. Furthermore, $\mathbf{v_1}$ is the vector that, when multiplied with the matrix A, strectches by a factor of $\sigma_\mathbf{1}$ and rotates so that it points along $\mathbf{u_1}$. Therefore columns of matrix $\mathbf{V}$ are the principal components of the matrix $\mathbf{A}$.

3 Algorithm Implementation and Development

The analysis primarily constituted five steps:

Data Collection and Formatting
A video was downloaded from the Youtube, where a person is walking around the city with a camera held on his hand. The video captured was relatively stable in all directions except in the vertical direction where frames were moving up and down because of walking motion. The video was loaded and each image was grayscaled and downscaled by a factor of 1/4 to obtain 270 x 480 frame.

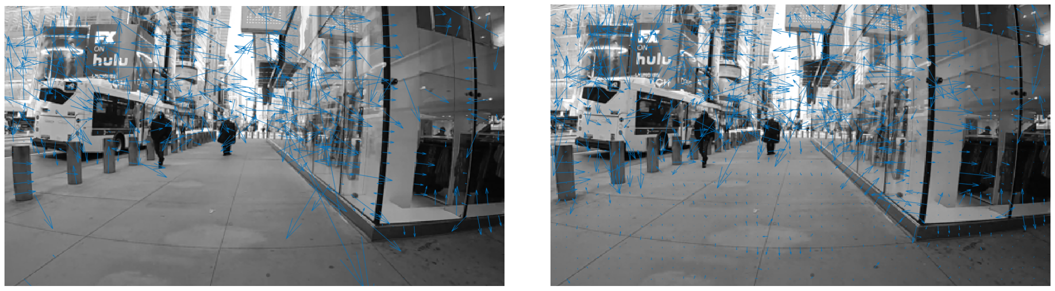
Optical Flow Estimation
MATLAB in-built command *estimateFlow* is used to estimate the flow matrix. Lucas-Kanade and Horn-Schunck methods are implemented using in-built objects *opticalFlowLK* and *opticalFlowHS* as command parameters. It returns a dense optical flow estimates (V_x, V_y) at each pixel.

Matrices Construction and SVD
Two matrices corresponding to the flow estimates alongs x-direction and y-direction are transformed into column vectors. Two matrices F_X and F_Y are constructed such that their columns represent flow estimate matrix of each frames along x and y directions respectively. Each matrices is normalized to zero mean. Economic SVD is implemented on these matrices using MATLAB in-built command *svd.*

Video Reconstruction
Each image frame is warped using a low-rank approximation of the optical flow estimates. While the initial intent was to identify singular modes associated with distortions in the image due to walking, it became clear after implementation that isolating those modes is not easy, especially when modes are two dimensional (u, v). Therefore reconstruction was done simply by taking low-rank approximations.

4 Computational Results

A long video is downloaded such that the analysis can be performed on different sections of the video. The video can be found on Youtube and is $\approx$ 35 mins long. After video is loaded and image is formatted, optical



(a) Lucas-Kanade method (b) Horn-Schunck method

Figure 1: Optical Flow estimates using MATLAB in-built function.

flow is estimated. Figure 1 (a) and (b) show those estimates for Lucas-Kanade and Horn-Schunck method. Horn-Schunck method gives better estimates as it does not perform thresholding (unlike LK method) and uses multiple iterations for estimation. Figure 2 shows how each pixel is moving in a particular frame. Since it is easy to track edges across two frames, we can see all edges in the scene having strong velocities.

By horzontally stacking each of the flow velocities vectors along x-direction, matrix F_X is constructed. Similarly by stacking flow velocities along y-direction, matrix F_Y is constructed. Rows of the matrix represent variables, while columns represent measurements. These matrices are then singularly decomposed. The semilogy plot of energy contained in each of the singualr modes shows that these modes are compact and contain low-energy. This makes sense since there are large number of variables (each pixel essentially is a variable).

Figure 3 shows the dominant mode that captures the horizontal and vertical motion of pixels in the scene, due to the motion of the camera. They are exactly what we would have expected. A horizontal motion is best captured by tracking the movement of features on vertical lines/structures. Similarly, a vertical motion is best captures by tracking features on the horizontal lines. Thus, these two modes represent the forward motion of the person with camera. The other 299 modes can represent numerous features like motion of the objects themselves in the video and noise. Figure 4 shows the most dominant modes in each direction.

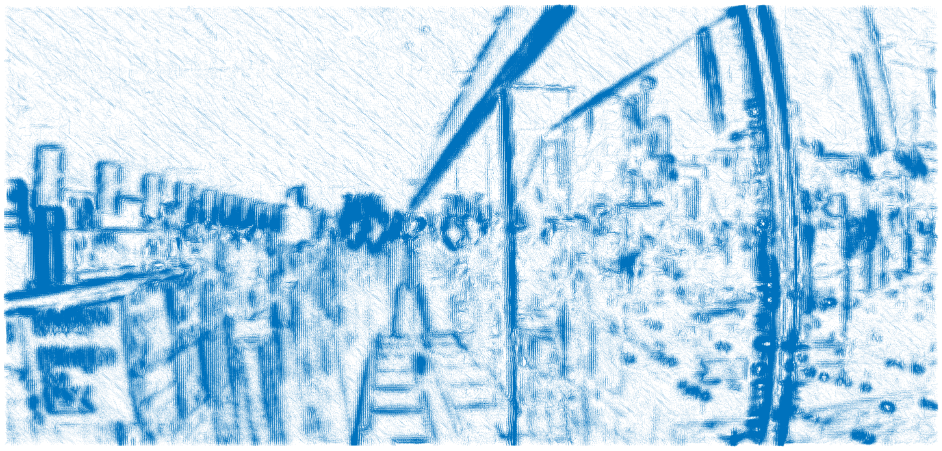
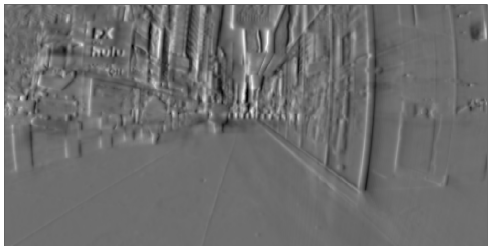
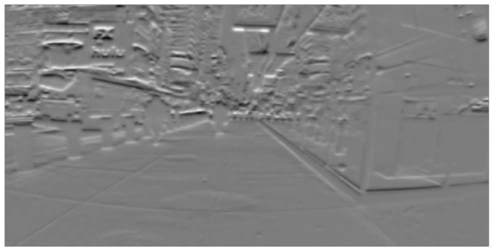


Figure 2: A vector field representing flow velocities at each pixel point between two consecutive frames.

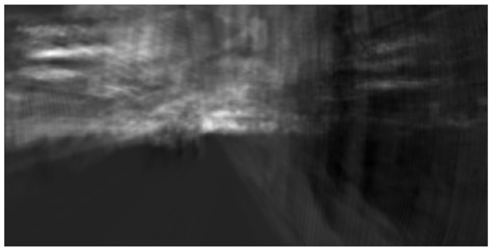


(a) 4th dominant mode along x-direction.

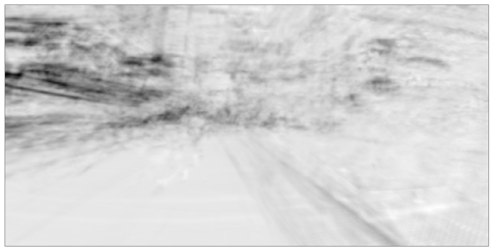


(b) 2nd dominant mode along y-direction.

Figure 3: Dominant modes that capture horizontal and vertical motion of objects in the camera. (relative motion)



(a) 1st dominant mode along x-direction.



(b) 1st dominant mode along y-direction.

Figure 4: Most dominant modes in the SVD.

The black region in figure (a) indicates 0 flow velocities. This coincides with the region in the image where buildings are and they do not move themselves. Thus this mode indicates the motion of objects itselfs like buses, cars and pedestrians. It is not clear what figure (b) is representing as the pattern is not obvious.

My initial intent was to similarly identify the mode associated with walking, but it seems there is no clear way to find that out. The problem itself is little ambigious because a pixel (object) can move up by either moving up itself, camera moving down or camera moving forward. The questions then is how do we distinguish those cases. On this paper, I was not able to figure that out.

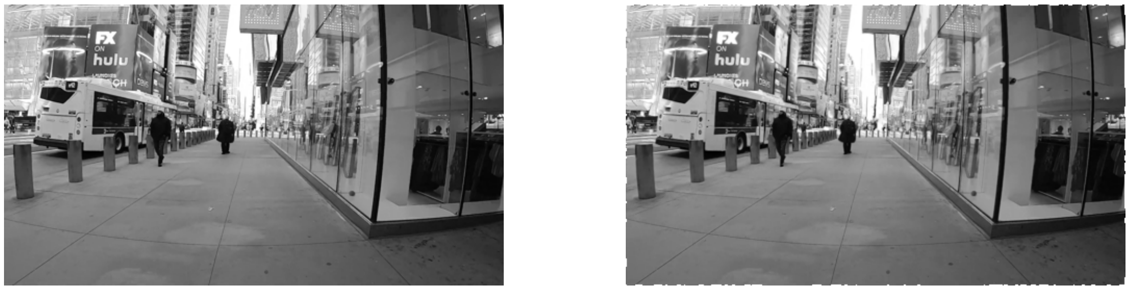
The next stage was to reconstruct the video by using a low-rank approximation of the flow velocity matrix $(F_X(r)$,$(F_Y(r))$.

$$F_X(r) = U_X[:, 1:r]S_X[1:r, 1:r]U_X[:, 1:r]' + \bar{X}$$
$$F_Y(r) = U_Y[:, 1:r]S_Y[1:r, 1:r]U_Y[:, 1:r]' + \bar{Y}$$

where $\bar{X}$ and $\bar{Y}$ are means of the flow velocities along x and y directions.

Each image was then warped using the MATLAB in-built 2D interpolation function *interp2*. The following relation was used to find the warping matrix.

$$X_p = X - f_X$$
$$Y_p = Y - f_Y$$



(a) Original Image (b) Reconstructed Image.

Figure 5: Image reconstruced with 20-rank approximation of flow velocity matrices.

where (X, Y) defines the mashgrid of pixel points in the image space and (f_x, f_Y) are flow velocity matrices of a frame. The approximations were decent, as seen in previous homeworks. Figure 5 shows the original and reconstructed image with first 20 principal modes of both SVD.

5 Summary and Conclusions

In this paper, Singular Value Decomposition was used to stabilize video captured with a hand held camera while the person is walking. The idea was to find the mode associated with walking and isolate that mode from the SVD. However, it became clear this was not as easy as it sounds. The paper discusses different principal modes that were captured and how we can study some of them and identify what they represent in the scene. Lucas-Kanade and Horn-Schunck methods of solving optical flow equations were introduced and implemented using MATLAB built-in functions. The video was reconstructed using a low-rank approximation of flow velocity matrices.

Appendix A MATLAB Functions

Some important MATLAB functions used during the implementation.

- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix A, such that A = U*S*V'.
- `B = repmat(A,r1,...,rN)` specifies a list of scalars, r1,..,rN, that describes how copies of A are arranged in each dimension. When A has N dimensions, the size of B is size(A).*[r1...rN]. For example, repmat([1 2; 3 4],2,3) returns a 4-by-6 matrix.
- `imagesc(C)` displays the data in array C as an image that uses the full range of colors in the colormap
- `movegui(f,position)` moves the figure f to the specified screen location.
- `p = uipanel` creates a panel in the current figure and returns the Panel object. If there is no figure available, MATLAB® calls the figure function to create one.
- `opticFlow = opticalFlowLK` returns an optical flow object that you can use to estimate the direction and speed of the moving objects in a video. The optical flow is estimated using the Lucas-Kanade method.
- `opticFlow = opticalFlowHS` returns an optical flow object that you can use to estimate the direction and speed of the moving objects in a video. The optical flow is estimated using the Horn-Schunck method.
- `flow = estimateFlow(opticFlow,I)` estimates optical flow between two consecutive video frames.
- `imwrite(A,filename)` writes image data A to the file specified by filename, inferring the file format from the extension. imwrite creates the new file in your current folder.
- `v = VideoReader(filename)` creates object v to read video data from the file named filename.
- `Vq = interp2(X,Y,V,Xq,Yq)` returns interpolated values of a function of two variables at specific query points using linear interpolation.
- `h = quiver(...)` A quiver plot displays velocity vectors as arrows with components (u,v) at the points (x,y).

Appendix B MATLAB Code

The code is published on the github repository AMATH-582 under Video Stabilization.

MATLAB code for Eigenfaces problem.

```
%% Clear Workspace
close all; clear variables; clc

%% Global Variables
Verbose = 0;

%% Load the video and construct the matrix
vidReader = VideoReader('video.mp4','CurrentTime',1130);

% Using Horn-Schunck method
opticFlow = opticalFlowHS;

% Plotting axis for optical flow
h = figure;
movegui(h);
hViewPanel = uipanel(h,'Position',[0 0 1 1],'Title','Plot of Optical Flow
   Vectors');
hPlot = axes(hViewPanel);
set(h,'DefaultLineLineWidth',10)

% Initialize matrices
FX = [];
FY = [];

nFrames = 0;
while hasFrame(vidReader) && nFrames < 300
    frameRGB = readFrame(vidReader);
    frameGray = rgb2gray(frameRGB);
    frameResize = imresize(frameGray, 0.25);
    flow = estimateFlow(opticFlow,frameResize);

    f_x = flow.Vx;
    f_y = flow.Vy;
    FX = [FX double(reshape(f_x, [],1))];
    FY = [FY double(reshape(f_y, [], 1))];

    imshow(frameResize)
    hold on
    plot(flow,'DecimationFactor',[10 10],'ScaleFactor',100,'Parent',hPlot)
        ;
    hold off
    pause(0.001)
    nFrames = nFrames + 1;
end

%% Singular Value Decomposition
% Zero-mean the matrices before SVD
X_mean = repmat(mean(FX,1),129600,1);
```

```
FX = FX-X_mean;
Y_mean = repmat(mean(FY,1),129600,1);
FY = FY-Y_mean;

[UX,SX,VX] = svd(FX, 'econ');
[UY,SY,VY] = svd(FY, 'econ');

%% Plot principal modes
figure()
for i=1
    imagesc(reshape(UY(:,i),270,480)); colormap gray;
    set(gca,'xtick',[],'ytick',[])
    pause(2);
end

% Plot energy content of each singular modes
sigma_y = diag(SY);
figure()
plot((sigma_y/sum(sigma_y))*100, 'r*')

%% Extra plotting for the paper
if Verbose
    VX_p = UX(:,1:10)*SX(1:10,1:10)*VX(:,1:10)';
    VY_p = UX(:,1:10)*SX(1:10,1:10)*VX(:,1:10)';

    VX_p = VX_p + X_mean;
    VY_p = VY_p + Y_mean;

    [X,Y] = meshgrid(1:480,1:270);
    X_p = X-reshape(VX_p(:,200), 270, 480);
    Y_p = Y-reshape(VY_p(:,200), 270, 480);

    % Plot final flow velocity after warping
    figure()
    q = quiver(X_p,Y_p);
    q.AutoScale = 'on';
    q.AutoScaleFactor = 2;

    % Plot flow field of 200th frame.
    figure()
    q3 = quiver(reshape(FX(:,200), 270, 480),...
        reshape(FY(:,200), 270, 480), 'linewidth', 5);
    q3.AutoScale = 'on';
    q3.AutoScaleFactor = 2;
    axis off

    % Plot 10-rank approximation of the flow field of 200th frame
    figure()
    q4 = quiver(reshape(VX_p(:,200), 270, 480),...
        reshape(VY_p(:,200), 270, 480));
    q4.AutoScale = 'on';
    q4.AutoScaleFactor = 2;
    axis off
end
```

```matlab
%% Reconstruct the video using low-rank approximation
% 20-rank approximation of flow vield matrices.
VX_p = UX(:,1:20)*SX(1:20,1:20)*VX(:,1:20)' + X_mean;
VY_p = UX(:,1:20)*SX(1:20,1:20)*VX(:,1:20)' + Y_mean;

[X,Y] = meshgrid(1:480,1:270);

vidReader = VideoReader('video.mp4','CurrentTime',1130);
nFrames = 1;

% Read each image frame
while hasFrame(vidReader) && nFrames <= 300
    frameRGB = readFrame(vidReader);
    frameGray = rgb2gray(frameRGB);
    frameResize = imresize(frameGray, 0.25);

    % Find previous pixel locations using flow velocity
    X_p = X-reshape(VX_p(:,nFrames), 270, 480);
    Y_p = Y-reshape(VY_p(:,nFrames), 270, 480);

    % Warp the image
    image = interp2(im2double(frameResize), X_p, Y_p);
    imshow(image)
    pause(0.1)

    % Save the image
    imwrite(image, ['./Result/image' num2str(nFrames) '.png'])
    nFrames = nFrames + 1;
end

%% Check reconstructed images
files = dir("./Result/image*");
for file=files'
    img = imread([file.folder '/' file.name]);
    imshow(img)
    pause(0.1)
end

%% Observe each column of the matrix
if Verbose
    figure()
    for i=1:300
        imagesc(reshape(FX(:,i),270,480)); colormap gray;
        set(gca,'xtick',[],'ytick',[])
        pause(2);
    end
end
```