Principle Component Analysis

Eileen Chang

February 21, 2020

Abstract

There are four different sets of videos that represent different oscillation scenarios. In each set, there are three videos that shoot from different locations and different angles but films at the same time. We are going to use the principle component analysis to convert three sets of the x and y coordinates at n points in time into the six principle components. We then project the data onto the principle component basis sets and compared them with original behavior.

1 Introduction and Overview

The principal component analysis is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The principal components are orthogonal because they are the eigenvectors of the covariance matrix, which is symmetric.
In this paper, we are going to analyze the four different cases, which are ideal cases, noisy case, horizontal displacement case, and the horizontal displacement and rotation case. Four tests are performed on an oscillating mass. The bucket is moved and is filmed by three different cameras that placed in different places. In the ideal case, the bucket is in the z-direction with simple harmonic motion. The noisy case is similar to the ideal case but having some noise in the video. The horizontal displacement case is the bucket released off-center so as to produce motion in the x-y plane as well as the z-direction. The last case, the horizontal displacement and rotation, is same as the previous case but the bucket will rotation. We are going to apply the principal component analysis in these videos to analyze the movement of the buckets in the videos.

2 Theoretical Background

The Singular Decomposition(SVD) is a method to decompose the matrix into following:

$$A = U\Sigma V^* \tag{1}$$

where U is an m*m unitary matrix, $\sum$ is an m*n rectangular diagonal matrix with non-negative numbers on the diagonal, and V is an n*n unitary matrix. The matrices U and V^* are rotational matrices and $\sum$ is a stretching matrix. The SVD allows every matrix to be diagonal if the proper bases for the domain and the range are used. The SVD is also a least-square fitting algorithm. It allows us to project the matrix on to the low-dimensional representations.

On of the primary application of SVD is the Principle Component Analysis(PCA). Like its name, the PCA will help our analysis the most important part of the data. If the data is 2d, then the PCA will help us find the line that fits the data almost perfectly. If the data is 3d, then the PCA will give us a plane. The PCA allows the random sets of data to be reduced to lower dimensions of dynamics without any underlying behavior. Be able to help us find the bounding box is a useful part of the PCA. The key to analyzing the data is to consider the covariance matrix, which is

$$Cx = \frac{1}{n-1} X * X' = \frac{1}{n-1} \Sigma^2, \tag{2}$$

where the size is m*m. The diagonal terms of Cx are the variances for particular measurements. The key idea behind the diagonalization is there exists an ideal basis in which the Cx can be written so that in the basis, all redundancies have been removed, and the largest variances of particular measurements are ordered.

3 Algorithm Implementation and Development

For each test, the algorithms that used to extract the position of bucket in the video are similar.

1. Load the three given .mat videos corresponding to different position of camera. And each .mat videos is saved in a 4D matrix.

2. Using the size command to find the size of each frame and the number of frames for each video.

3. Convert the videos from RGB to grayscale by using the rgb2gray command.

4. Set all the pixels to zero except the small window, so we can isolate the portion of the video where the movement of bucket occurs.

5. Since the bucket has the flashlight attached to the top of it, we use max command to search the window for the maximum intensity of pixel.

6. Convert the index that we obtained in previous step to (x,y) by using the command ind2sub. We then saved each x and y to a vector X and a vector Y.

7. Each video has different time. In order to make them having the same time, we find the shortest video among them and cut other videos to make them all have the same time length.

8. Formed a matrix A that constructed by X1, Y1, X2, Y2, X3, Y3.

9. Compute the size of the matrix A.

10. Compute the mean for each row in the matrix A and then subtract the mean.

11. Using the matrix A to do the singular value decomposition by using the command svd.

12. Take the transpose of eigenvectors and times the matrix A to produce the principal components projection.

4 Computational Results

4.1 Test 1

Test 1 is the ideal case. In this case, the entire motion is in the z-direction with a simple harmonic motion. Figure 1 is the trajectories in x-direction and y-direction. Each row represents different cameras. From figure 1, we can tell that x-direction is kind of stable. However, in the y-direction, the wave is kind of unstable. It goes up and down. From figure 2, we can tell that there is only one principal component which is the 91.78% energy capturing. The others are way lower compared with the first principle component. Thus, it fits the video very well since there is only a bucket oscillating in one direction.

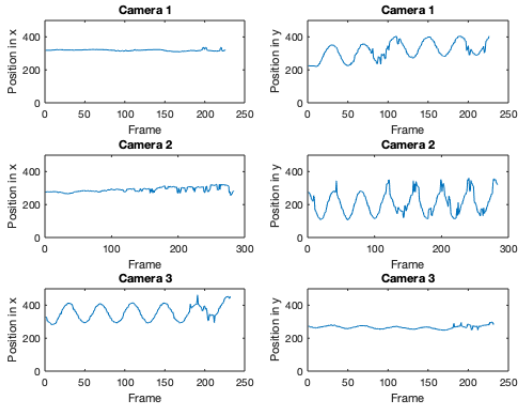

Figure 1: The x direction and y direction from three different cameras (Ideal)

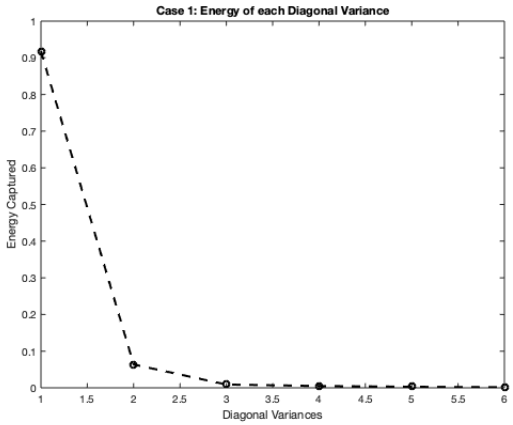

Figure 2: Principle Component (Ideal)

4.2 Test 2

Test 2 is similar to the test 1 experiment, but this time, introduce camera shake into the video recording. From figure 3, we can see that compare with figure 1, the wavelets are kind of messy than the first case because the camera shakes while filming. Figure 4 is a principal component. We can see that there is not only one principal component domains all the video since the second principal component has a somehow high value of energy, which is 0.195. So, we have to consider two principal components for this system. Even though there is only a camera shaking difference between test 1 and test 2, we have different principal components for these two systems. Thus, noise indeed will throw off some of the principal component analysis calculations.

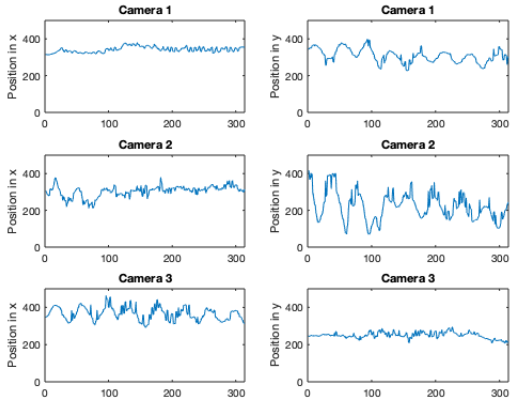

Figure 3: The x direction and y direction from three different cameras (Noisy)

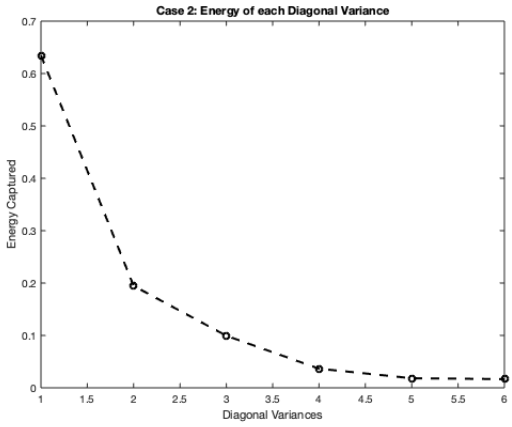

Figure 4: Principle component (Noisy)

4.3 Test 3

Test 3 is the case of horizontal displacement. In this case, the mass is released off-center so as to produce the motion in the x-y plane as well as the z-direction. Thus, there is both a pendulum motion and simple harmonic oscillations. In figure 5, the y-direction seems more smooth than the previous two cases. It doesn't have high amplitudes. And the x-direction looks kind of periodic. So, we can tell that the bucket is mainly moving horizontally. Moreover, we can see that there are four principal components captured the energy of this system since four of them have pretty high values.

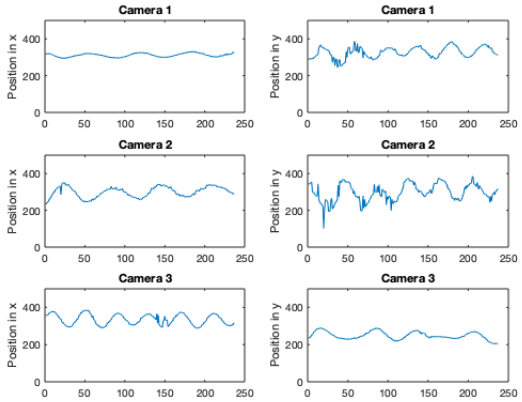

Figure 5: The x direction and y direction from three different cameras (Horizontal displacement)

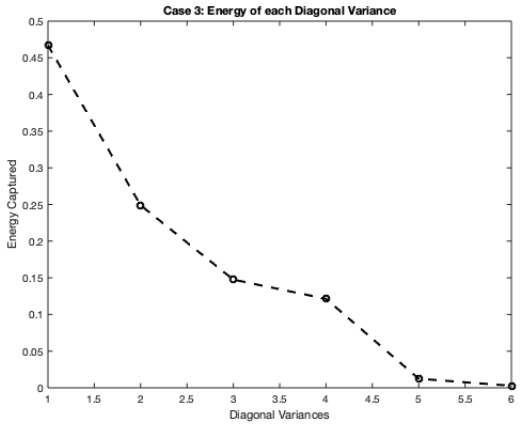

Figure 6: Principle component (Horizontal displacement)

4.4 Test 4

Test 4 is the case that the mass is released off-center and rotates so as to produce motion in the x-y plane, rotation, and motion in the z-direction. Compare with the test 3 case, in both camera three, the x-direction in test 4 has higher frequency since the wavelength, in this case, is shorter than the previous case. And in both camera two, the test 4 case has a messier wavelet compare with the test 3 case. Thus, we can tell that there is rotation in the x-direction. From figure 8, we can see that there are three principal components in the system that have high energy captured. It seems that the PCA captures the multi-dimensional nature of the horizontal displacement and the rotation.

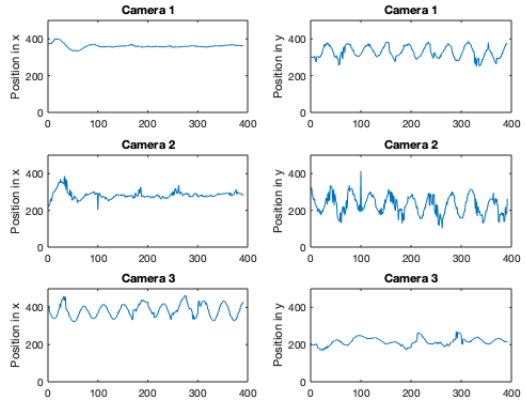

Figure 7: The x direction and y direction from three different cameras (Horizontal displacement and rotation)

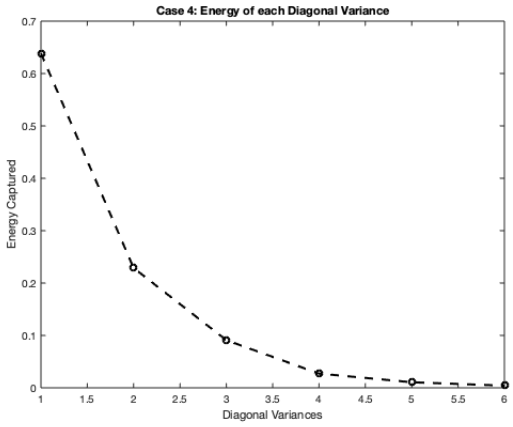

Figure 8: Principle component (orizontal displacement and rotation)

5 Summary and Conclusions

The technique of Principal Component Analysis is applied to videos of an oscillating mass. It is a useful tool. PCA allows us to see how many significant, orthonormal, principal components existed. From the figures of the principal component in each test, we can tell that the first test is the best case. It only has one principle component which is the "ideal case". We observe that our projections are accurate to some extent. Doing the principal component analysis can help us put out the most important information and get rid of the unuseful things.

6 Appendix A.

1. load
   - Load variables from file into workspace
   - s = load()
2. rbg2gray
   - Convert RGB image or colormap to grayscale
   - I = rgb2gray(RGB)
3. svd
   - Singular value decomposition
   - $[U, S, V] = svd(A)$ performs a singular value decomposition of matrix A, such that A = U*S*V'.

4. ind2sub
    - Convert linear indices to subscripts
    - $[row, col] = ind2sub(sz, ind)$ returns the arrays row and col containing the equivalent row and column subscripts corresponding to the linear indices ind for a matrix of size sz.
5. frame2im
    - Return image data associated with movie frame.
    - RGB = frame2im(F) returns the truecolor (RGB) image from the single movie frame F.
6. diag
    - Create diagonal matrix or get diagonal elements of matrix.
    - D = diag(v) returns a square diagonal matrix with the elements of vector v on the main diagonal.
7. size
    - Array size
    - sz = size(A) returns a row vector whose elements are the lengths of the corresponding dimensions of A.
8. max
    - Find the maxi value in the vector/matrix.
    - M = max(A) will return the max value in array A.
9. abs
    - Return an absolute value.
    - A = abs(a).
10. min
    - Minimum elements of an array.
    - M = min(A) returns the minimum elements of an array.
11. plot
    - 2-D line plot
    - plot(X,Y) creates a 2-D line plot of the data in Y versus the corresponding values in X.
12. sort
    - Sort array elements
    - B = sort(A) sorts the elements of A in ascending order.

7 Appendix B.

This is the code for the test 1. Other tests have the similar code.

```
clear all; clc; close all;

%loading videos
load('cam1_1.mat');
load('cam2_1.mat');
load('cam3_1.mat');

%find numbers of frame
numFrames1 = size(vidFrames1_1,4);
numFrames2 = size(vidFrames2_1,4);
numFrames3 = size(vidFrames3_1,4);
maxFrames = max([numFrames1 numFrames2 numFrames3]);

%find size of frame
[m1,n1] = size(vidFrames1_1(:,:,1,1));
[m2,n2] = size(vidFrames2_1(:,:,1,1));
[m3,n3] = size(vidFrames3_1(:,:,1,1));

for k = 1:maxFrames
    if k <= numFrames1
        mov1(k).cdata = vidFrames1_1(:,:,:,k);
        mov1(k).colormap = [];
    end
    if k <= numFrames2
        mov2(k).cdata = vidFrames2_1(:,:,:,k);
        mov2(k).colormap = [];
    end
    if k <= numFrames3
        mov3(k).cdata = vidFrames3_1(:,:,:,k);
        mov3(k).colormap = [];
    end
end

X1=[];X2=[];X3=[];Y1=[];Y2=[];Y3=[];

for i = 1:maxFrames
    if i <= numFrames1
        abw = rgb2gray(frame2im(mov1(i)));
        abw(:,1:320) = 0;
        abw(:,380:end) = 0;
        abw(1:200,:) = 0;
```

```
        [Max, Ind] = max(abw(:));
        [y1 x1] = ind2sub(size(abw), Ind);
        X1 = [X1 x1];
        Y1 = [Y1 y1];
    end
    if i <= numFrames2
        abw = rgb2gray(frame2im(mov2(i)));
        abw(:,1:260) = 0;
        abw(:,330:end) = 0;
        [Max, Ind] = max(abw(:));
        [y2 x2] = ind2sub(size(abw), Ind);
        X2 = [X2 x2];
        Y2 = [Y2 y2];
    end
    if i <= numFrames3
        abw = rgb2gray(frame2im(mov3(i)));
        abw(1:250,:) = 0;
        abw(310:end,:) = 0;
        abw(:, 1:260) = 0;
        [Max, Ind] = max(abw(:));
        [y3 x3] = ind2sub(size(abw), Ind);
        X3 = [X3 x3];
        Y3 = [Y3 y3];
    end
end

figure(1)
subplot(3,2,1)
plot(X1);xlabel('Frame');ylabel('Position in x');ylim([0 500]);title('Camera 1')

subplot(3,2,2)
plot(Y1);xlabel('Frame');ylabel('Position in y');ylim([0 500]);title('Camera 1')

subplot(3,2,3)
plot(X2);xlabel('Frame');ylabel('Position in x');ylim([0 500]);title('Camera 2')

subplot(3,2,4)
plot(Y2);xlabel('Frame');ylabel('Position in y');ylim([0 500]);title('Camera 2')

subplot(3,2,5)
plot(X3);xlabel('Frame');ylabel('Position in x');ylim([0 500]);title('Camera 3')

subplot(3,2,6)
plot(Y3);xlabel('Frame');ylabel('Position in y');ylim([0 500]);title('Camera 3')

[Min1 I1]=min(X1(1:50)); X1=X1(I1:I1+200); Y1=Y1(I1:I1+200);
```

```
[Min2 I2]=min(X2(1:50)); X2=X2(I2:I2+200); Y2=Y2(I2:I2+200);
[Min3 I3]=min(X3(1:50)); Y3=Y3(I3:I3+200); X3=X3(I3:I3+200);


A = [X1;Y1;X2;Y2;X3;Y3];
[m,n]=size(A);
mn=mean(A,2);
A=A-repmat(mn,1,n);

[u,s,v] = svd(A);

Y = u'*A;

figure(2)
plot(1:6, (diag(s).^2)/sum(diag(s).^2), 'ko--', 'Linewidth', 2);
title("Case 1: Energy of each Diagonal Variance");
xlabel("Diagonal Variances"); ylabel("Energy Captured");
```