# AMATH 482 Homework 3

Sandy Yang

Feb 22, 2021

**Abstract**

In this assignment, we will use Singular Value Decomposition (SVD) and Principal Component Analysis (PCA) to analyze videos of a hanging mass taken by cameras from three different cameras. Four different cases has been studied: : ideal case which is just simple harmonic oscillation with ups and downs, noisy case, horizontal displacement, and horizontal displacement with rotation.

## 1 Introduction and Overview

The majority of the objects in our daily life are moving all the time. How is this object moving and what is the moving trajectories are the things that we care about. However, for different people from different angles observing the movement, they may have different answers toward one movement. In order to figure out the truth, it is the same as solving a case. We need to collect data from various directions and find out the most significant ones, which are the truth we are trying to find.

In this assignment, our task is to analyze the movement of the mass. Three cameras are given from different angle and we also have 4 different cases have to study for each camera: ideal case, noisy case, horizontal displacement, and horizontal displacement and rotation. What we will do is to find out the moving trajectories of the mass, saving its coordinate, and using PCA and SVD on three set of coordinates in order to find the principle component.

## 2 Theoretical Background

### 2.1 The Singular Value Decomposition (SVD)

All matrices can be describe as stretching and rotation of a particular vector. A singular value decomposition (SVD) is a factorization of matrix into a number of constitutive components.

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

Where U is a unitary matrix, $\Sigma$ is a rectangular diagonal matrix and V is also a unitary matrix. The matrices U and V are rotational matrices and matrix $\Sigma$ is a stretching matrix. Performing SVD to a matrix remove redundancy and identify the signals with maximal variance. SVD is tool of choice for data analysis and dimensional reduction. In fact, the SVD diagonalizes and each singular directions captures as much energy as possible as measured by the singular values $\sigma_j$.

#### 2.1.1 Principal Component Analysis (PCA)

PCA produces the eigenvalue decomposition and the projection of the original data onto principle component basis using the diagonalization of the matrix by performing SVD. Suppose we have 2 measurement from 3 different cases and we make vectors of all the position at each time, we will get the coordinate matrix:

$$X = \begin{bmatrix} x_a \\ y_a \\ x_b \\ y_b \\ x_c \\ y_c \end{bmatrix} \tag{2}$$

The coordinate matrix has dimension m by n, where m is the number of measurements and n represents the number of data points that were taken for each measurement. We can determine how much redundancy exists in this sets of measurements by computing the m by m covariance matrix.

$$C_X = \frac{1}{n-1} X X^T \tag{3}$$

If all the trajectory we captured are independent interesting dynamics then $C_X$ would have large diagonal entries and very small off-diagonal entries (because of the low redundancy between measurements). This is also the goal of PCA: find a basis in which $C_x$ is diagonal. To achieve this, We use the SVD since it can be applied on any matrix. Each singular direction in the SVD captures as much energy as possible, which is measured by the singular values $\sigma_j$. Suppose we have X have the SVD($X = U\Sigma V^*$), we can project the data onto the left singular values and call the result $Y = U^T X$. Then, the covariance of Y is:

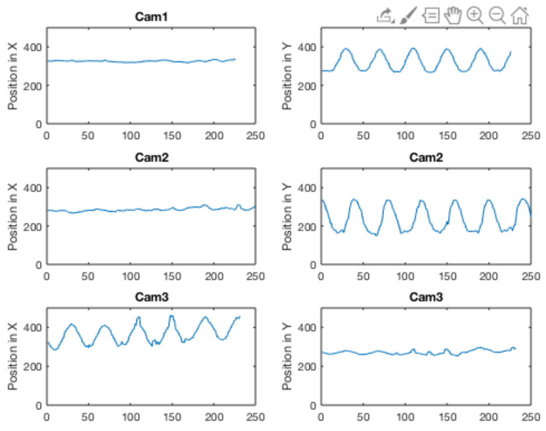
$$C_Y = \frac{1}{n-1} Y Y^T = \frac{1}{n-1}(U^T X)(U^T X)^T = \frac{1}{n-1} U^T (X X^T) U = \frac{1}{n-1} U^T U \Sigma^2 U^T U = \frac{1}{n-1}\Sigma^2 \tag{4}$$

Thus the basis formed by the left singular vectors of X is the ideal one in which to express the data stored in X in the sense that this representation leads to a diagonal covariance matrix. To determine the directions along which important dynamics occur, we can consider the energy in each principal component direction, which are the singular values $\sigma_j$. Typically, if redundancy is present in the measurements, X will have only a few large singular values, and the rest will be minuscule in comparison.

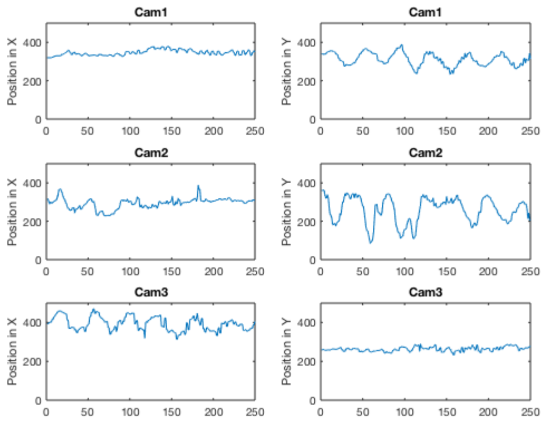
## 3 Algorithm Implementation and Development

The algorithm for extracting original data and data after PCA is the same for the four tests of the project.

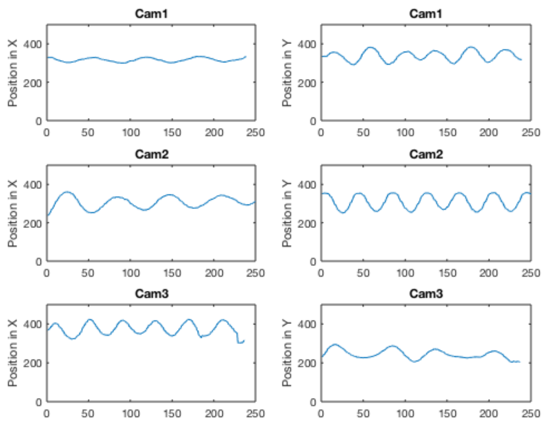
- `Load camera mat files:` For each file loaded, the dimension is $m \times n \times p \times t$, where m and n is the size for each image, p is the number of rbg layers and t is the number of images in this video.
- `Convert RGB data to grayscale:` In order to trace the object in each image, I change the color images into gray-scale by using the command "rgb2gray" inside the for loop.
- `Isolate image into small portion:` Since there are many places where the color(in grayscale) are still very similar, we can blacked out the background environment, leaving a small window where the target object can be fully focused and clean out the noises in the surroundings and avoid distracting. We can accomplish this by setting all other surrounding pixels to zero in the forloop. Then, we can set a threshold where we found all locations where the value was greater than around 250 (we need to adjust the number due to different case).
- `Located the object in the frame:` Then, in the forloop, we can use "find" and "ind2sub" commands to locate the coordinates of the mass, which can help us detect the coordinates of the target mass. We then averaged out the coordinates, and store this to our data array.
- `Plot out the trajectory` We can plot out the trajectory for each camera so later we can have a better understanding on how the PCA perform.



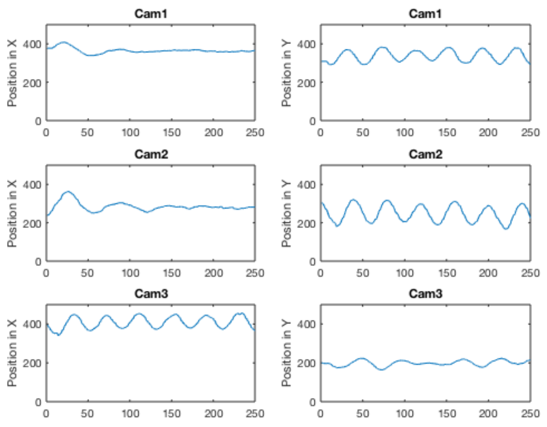
(a) Ideal Case



(b) Noisy Case



(c) Horizontal Displacement



(d) Horizontal Displacement and Rotation

Figure 1: X and Y movement for all camera at 4 cases

- `Make coordinate matrix` After getting coordinate for each camera, construct the coordinate matrix, such that we had 6 rows corresponding to X and Y coordinates taken from 3 videos, and a number of columns equal to the lowest frame number. Since each camera has different number of image, we set other cameras length to be the same length as the shortest one.

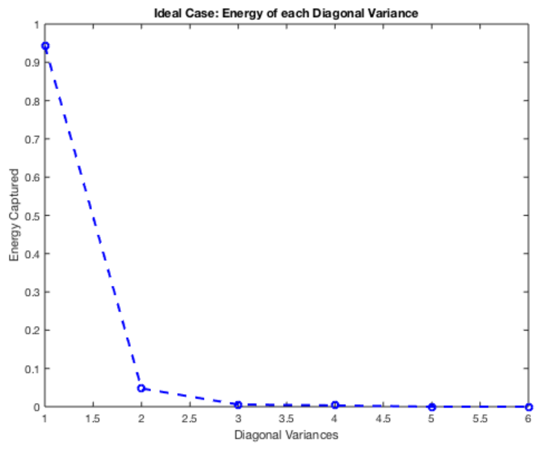
- `Perform SVD:` With this coordinate matrix, we can subtract the mean of each row from the data to center the data, and then used took the svd of the transpose of our data matrix, and divided by square root of $n - 1$, using `svd(data'/sqrt(n-1)` to get three matrices, [u,s,v]. Extracted the values of the diagonal matrix s and squared them to get our variances for each principal component.

- `Graph:` Plot the normalized variances (divided by the sum) and see which principal components occupied the high energy and then plotted out that projections on the significant principal component orthonormal bases to reconstruct our observed data. We repeated this procedure for all four sets of data.

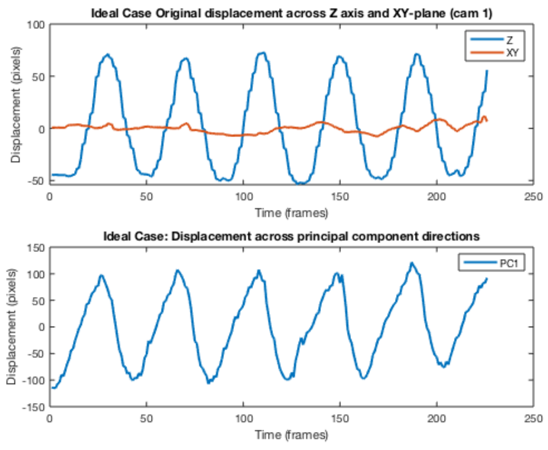
## 4 Computational Results

### 4.1 For case 1:

Based on figure 2a, there is only one principal component since it has a really high percent of energy capturing. The rest of the components had very low energy, which make sense since our target mass is oscillating in one single direction (See Figure 1). Moreover, our transformation of the first principal component resulted in accurate data. We also see that our projection onto the first principal component basis results in very similar data to what we experimentally measured. This shows that this system can be accurately reproduced and represented by one principal component.



(a) variances of principal components

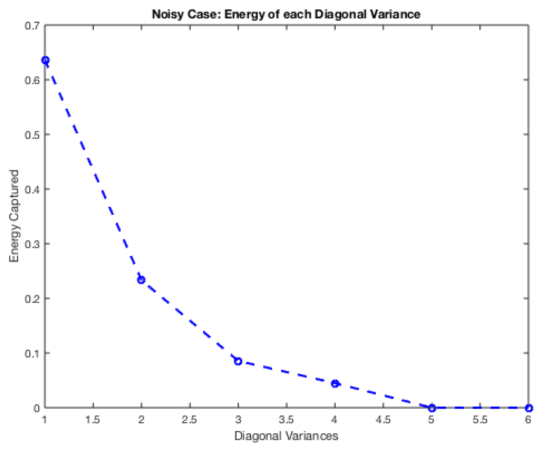


(b) z displacement and new basis

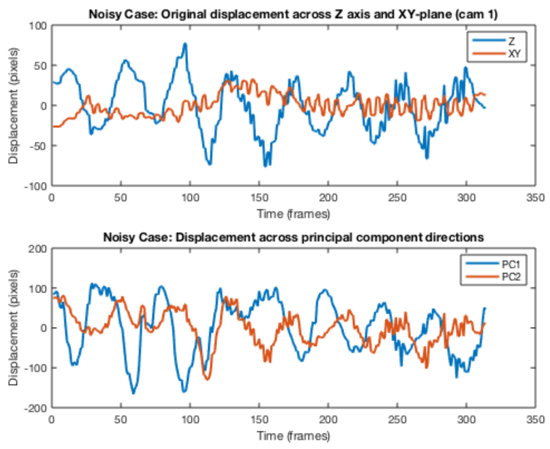
Figure 2: Case 1: Ideal Case

### 4.2 For case 2:

Based on figure 3a, we may have to consider two principal to be significant since the second diagonal variance also look a little high. Thus, let's try to plot those two principal components. It's clear that the second principal component is due to the noise in the data and it has thrown off some of our PCA calculations. This is also true in our projection to the principal component basis. However, although there is lots of noise in both systems, there is clearly oscillatory behavior.



(a) variances of principal components

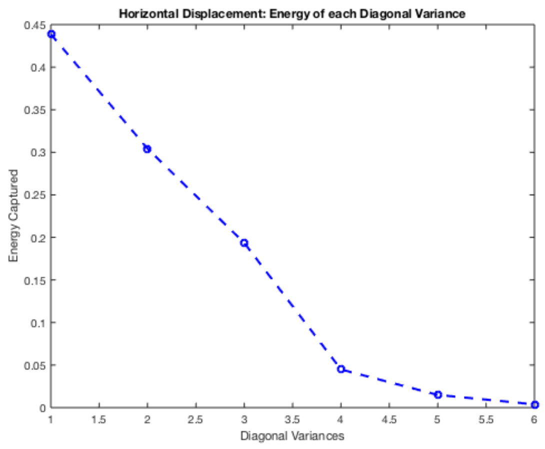


(b) z displacement and new basis

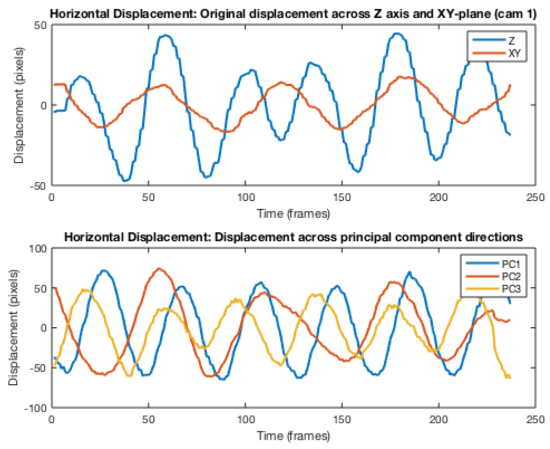
Figure 3: Case 2: Niosy Case

## 4.3 For case 3:

Based on figure 4a, we see there are 3 principal components that seem to capture significant amounts of energy relative to each other (the fourth one is considerable small so we ignore it). Although there is a large drop off from the first principal component to the last, they are all relatively close together.



(a) variances of principal components

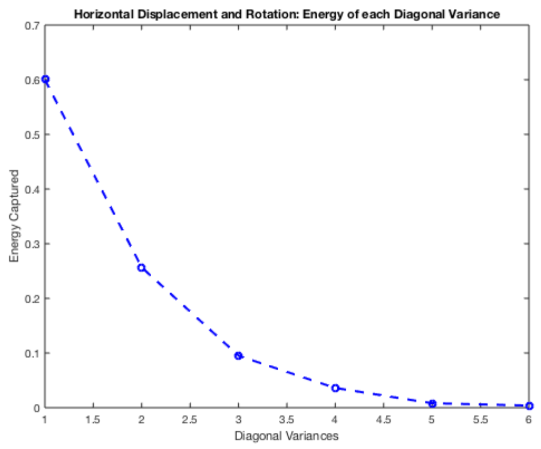


(b) z displacement and new basis

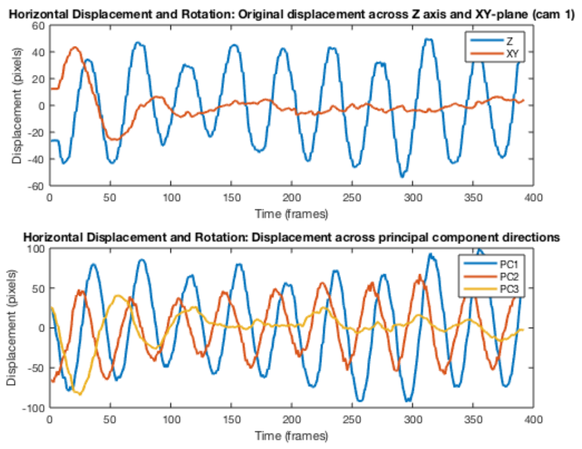
Figure 4: Case 3: Horizontal Displacement

## 4.4 For case 4:

Based on figure 5a, there seem to be 3 significant principal components that have high energy captured. This make sense since, in this case, we have both the oscillation and rotation at the same time. It seems that PCA has captured the multi-dimensional behavior of the horizontal displacement and rotation, as there is both a pendulum nature, as well as simple harmonic motion.



(a) variances of principal components



(b) z displacement and new basis

Figure 5: Case 4: Horizontal Displacement and Rotation

## 5 Summary and Conclusions

To find the most significant components from a several set of data recording the same target/motion, PCA is considered to be really useful! Doing eigenvalue decomposition on the covariance matrix, we can get a diagonal matrix composed by the eigenvalues on the diagonal and the corresponding matrix composed by eigenvectors. The eigenvector matrix will be our principle component basis. Then, we can multiply the eigenvector matrix with the original matrix and get a projection of the original matrix on the principle component basis. Then we will what the original object or movement looks like. In short, PCA is very beneficial to get most important information and clean up redundancy.

## Appendix A MATLAB Functions

- `rgb2gray`: Turn image from rgb to gray-scale with black and white.
- `find(A)`: Find out the value and index of value that satisfy the given condition in a matrix.
- `[i,j,k] = ind2sub(siz,IND)`: Given a index IND, it returns the i, j, and k indeces, which can then be translated into spacial or frequency domain coordinates. We used this command to find the matrix coordinates of our bright spots of our black-white frames in the videos, which corresponded to the light on the target mass.
- `mean(A)`:: Returns the mean of the vector A. We used this to average the coordinates above the threshold value to locate the object.
- `diag(A)`: Returns a column vector of the diagonal values of a matrix.
- `size(A)`: Returns the dimensions of the matrix A.

## Appendix B MATLAB Code

The code for case 1 to 4 are very similar, the only difference is the number setting for threshold and the index number when filtering the image, so I only put the code for the first case.

```
close all; clear all; clc
%%
load('cam1_1.mat');
load('cam2_1.mat');
load('cam3_1.mat');

%%
% get the dimension of loaded file
% a11 and b11 = size of image
% c11 means layers of rgm image
% d11 = number of images in the vids
[a1, b1 , c1 , d1 ] = size(vidFrames1_1) ;

% get the basic idea of where the osscilation is
% use for creating small window
% imshow(rgb2gray(vidFrames1_1(:,:,:,1)))

XY1 =[];
for i = 1:d1

    %change each color image into gray-scale
   img = rgb2gray(vidFrames1_1(:,:,:,i));

   % blacked out the background environmen
   % leave the small window
   img(:,1:250) = 0;
   img(:,450:end) = 0;
   img(1:202,:) = 0;
   img(440:end,:) = 0;

   %imshow(img)

   % locate the coordinates of the mass
   thresh = img(:) > 250;
   indeces = find(thresh);
   [y, x] = ind2sub(size(img),indeces);
   XY1 = [XY1; mean(x), mean(y)];

end

%% for cam_2
[a2, b2 , c2 , d2 ] = size(vidFrames2_1);
```

```
% get the basic idea of where the osscilation is
% use for creating small window
% imshow(rgb2gray(vidFrames2_1(:,:,:,1)))

XY2 = [];
for i = 1:d2

    %change each color image into gray-scale
   img = rgb2gray(vidFrames2_1(:,:,:,i));

   % blacked out the background environmen
   % leave the small window
   img(:,1:240) = 0;
   img(:,350:end) = 0;
   img(370:end,:) = 0;
   %imshow(img)

   % locate the coordinates of the mass
   thresh = img(:) > 250;
   indeces = find(thresh);
   [y, x] = ind2sub(size(img),indeces);
   XY2 = [XY2; mean(x), mean(y)];

end

%% for cam_3
[a3, b3 , c3 , d3 ] = size(vidFrames3_1);

% get the basic idea of where the osscilation is
% use for creating small window
% imshow(rgb2gray(vidFrames3_1(:,:,:,1)))

XY3 = [];
for i = 1:d3

   %change each color image into gray-scale
   img = rgb2gray(vidFrames3_1(:,:,:,i));

   % blacked out the background environmen
   % leave the small window
   img(:,1:250) = 0;
   img(:,500:end) = 0;
   img(1:230,:) = 0;
   img(336:end,:) = 0;
   %imshow(img)

   % locate the coordinates of the mass
   thresh = img(:) > 247;
   indeces = find(thresh);
   [y, x] = ind2sub(size(img),indeces);
   XY3 = [XY3; mean(x), mean(y)];

end

%% Plot the result
figure(1)
subplot(3,2,1)
plot(XY1(:,1))
ylabel('Position in X')
ylim([0, 500])
xlim([0, 250])
title('Cam1')

subplot(3,2,2)
plot(XY1(:,2))
ylabel('Position in Y')
ylim([0, 500])
xlim([0, 250])
```

```matlab
title('Cam1')

subplot(3,2,3)
plot(XY2(:,1))
ylabel('Position in X')
ylim([0, 500])
xlim([0, 250])
title('Cam2')

subplot(3,2,4)
plot(XY2(:,2))
ylabel('Position in Y')
ylim([0, 500])
xlim([0, 250])
title('Cam2')

subplot(3,2,5)
plot(XY3(:,1))
ylabel('Position in X')
ylim([0, 500])
xlim([0, 250])
title('Cam3')

subplot(3,2,6)
plot(XY3(:,2))
ylabel('Position in Y')
ylim([0, 500])
xlim([0, 250])
title('Cam3')

%%
% to do SDV, we have to make a matrix in the same size
% so we cut the other two matrixs to the smallest one
min_len = min([length(XY1(:,1)), length(XY2(:,1)), length(XY3(:,1))]);

XY1 = XY1(1:min_len,:);
XY2 = XY2(1:min_len,:);
XY3 = XY3(1:min_len,:);

alldata = [XY1';XY2';XY3'];

%%
[m,n]=size(alldata); % compute data size
mn=mean(alldata,2); % compute mean for each row
alldata=alldata-repmat(mn,1,n); % subtract mean

[u,s,v]=svd(alldata'/sqrt(n-1)); % perform the SVD
lambda=diag(s).^2; % produce diagonal variances

Y= alldata' * v; % produce the principal components projection
sig=diag(s);

%%
figure()
plot(1:6, lambda/sum(lambda), 'bo--', 'Linewidth', 2);
title("Ideal Case: Energy of each Diagonal Variance");
xlabel("Diagonal Variances"); ylabel("Energy Captured");

figure()
subplot(2,1,1)
plot(1:min_len, alldata(2,:),1:min_len, alldata(1,:), 'Linewidth', 2)
ylabel("Displacement (pixels)"); xlabel("Time (frames)");
title("Ideal Case Original displacement across Z axis and XY-plane (cam 1)");
legend("Z", "XY")

subplot(2,1,2)
plot(1:min_len, Y(:,1),'Linewidth', 2)
ylabel("Displacement (pixels)"); xlabel("Time (frames)");
```

```
title("Ideal Case: Displacement across principal component directions");
legend("PC1")
```