HW3 : PCA

Author : Richy Yu

Abstract

This paper involves the use of principal component analysis. The purpose is simply to do some experiment with PCA, and discover its practical usefulness and the effects of noise on the PCA algorithms. The programming software used is MATLAB.

Introduction and Overview:

We are given set of videos taken from different cameras. The video is about oscillatory situations, but we do not want to use the governing equation from physics to solve this problem. Instead, we want to use the principal component analysis to analyze the videos and do some experiments. There are four situations in total, and they are ideal case, noisy case, horizontal displacement and horizontal displacement and rotation. In ideal case, we only consider a small displacement of the mass in the z direction. In the noisy case, we shake the camera to introduce noise. In the horizontal displacement, the mass is released off center so as to produce motion in the x-y plain as well as the z direction. In the horizontal displacement and rotation, the mass is released off-center and rotates. For each of these four cases, we use three different cameras to record the experiment. Since these cameras record the same experiment from different locations, there are some redundancies in the recording. We will do principal component analysis to experiment with these videos.

Theoretical Background

The singular value decomposition of a matrix A is defined as

$$\mathrm{A} = U\Sigma V^*, \qquad (\mathrm{e}.1)$$

Where U and V are unitary and Σ is diagonal. Diagonal values of Σ is ordered from largest to smallest. The following computations are extracted from our course notes to compute the singular value decomposition.

$$\begin{aligned} A^T A &= (U\Sigma V^*)^T (U\Sigma V^*) \\ &= V\Sigma U^* U\Sigma V^* \\ &= V\Sigma^2 V^* \end{aligned} \qquad (\mathrm{e}.2)$$

$$\begin{aligned} AA^T &= (U\Sigma V^*)(U\Sigma V^*)^T \\ &= U\Sigma V^* V\Sigma U^* \\ &= U\Sigma^2 U^* \end{aligned} \qquad (\mathrm{e}.3)$$

Multiplying (e.3) and (e.4) on the right by V and U respectively gives the two self-consistent eigenvalue problems

$$A^T AV = V\sum^2 \qquad (\mathrm{e}.4)$$

$$AA^T U = U\sum^2 \qquad (\mathrm{e}.5)$$

Thus if the normalized eigenvectors are found for these two equations, then the orthonormal basis vectors are produced for U and V. Likewise, the square root of the eigenvalues of these equations produces the singular values σ_j .

SVD is used for principle component analysis. Principle component analysis is to produce low-dimensional reductions of the dynamics and behavior when governing equations are not known.

Covariance matrix is defined as

$$C_X = \frac{1}{n-1} XX^T. \qquad (\mathrm{e}.6)$$

In MATLAB, we will use the function cov(X) to compute the covariance matrix. Covariance matrix tells the covariance between pairs of random variables and store these values in a matrix. The values on the diagonal are the variances of the variables. If two variables have large covariance, then they are dependent, and we say that they are redundant. If the diagonal values are large, we say that the variables corresponding to large values are more interesting.

Algorithm Implementation and Development

Since four videos all have three cameras, and they only differ in terms of the experiment itself, the process of the analysis is the same. The first step is to load the videos into MATLAB. Then we use the size function to find the number of frames of a video. We use a filter to remove the irrelevant information, so that we can focus on the motion of the object. Filter is a matrix of ones and zeros, and the size of the matrix is the same as the size of the frames of the video, so that scalar product could be performed. By trials and errors, the area of focus is found to be 90-400 X 230-360. We track the object by the light, which is the max value of the matrix.

Than we can find the location where the maximum occurs. We save the location of the object at each time frame.

Three cameras produce different number of time frames. In order to sync them, we pick the shortest video, and trim the longer videos, so that they match on the lowest y coordinate. Then three data are joined into a single matrix, which has six rows. This matrix matches the matrix we learnt in class in terms of the size, so that we could perform the decomposition and do the analysis. Then we subtract the mean of each row from the row to standardize, and divide the result by square root of n – 1. Then we can take the singular value decomposition of the result and get the diagonal variance and principle components projection. Then we plot the diagonal variance and projections.

Computational Results

Case one :

We only found one principal component which is at 88.89% energy capturing, as in figure 1. The next component is at 10% energy capturing. Since in this case, the mass is moving in one direction, having one principle component is expected. Also, our transformation of the principle component produce accurate oscillation graph, which is expected.

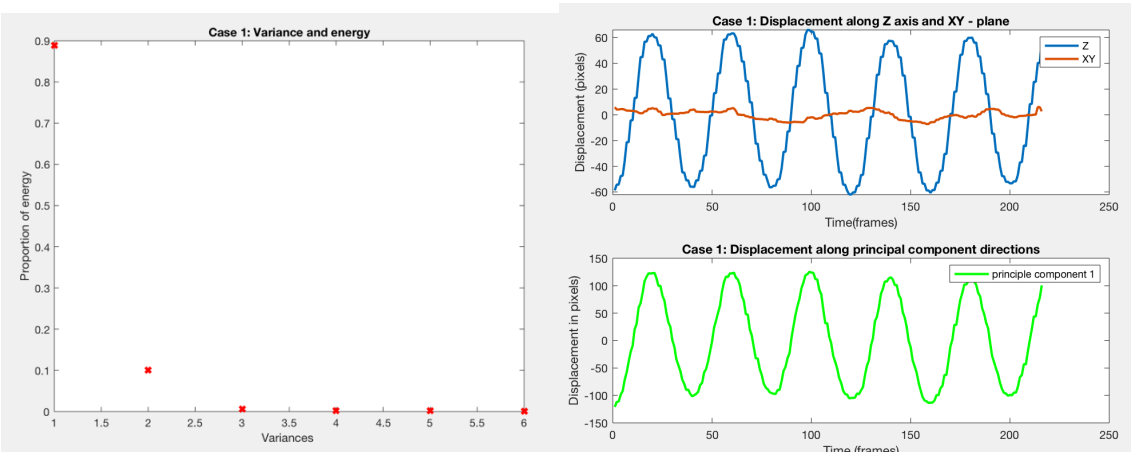

Figure 1 : This shows the principle components and the displacements of the object in two frames in case 1

Case two :

In this case, the first principle component has 64.29% energy capturing. The second component has less then 20% energy capturing. I consider that in second case, there are two principle components, because adding the shakes of camera introduces more noise to the movement of the object. As shown in figure 2, the movement of the object is similar to that in the first case, we get a oscillation motion, and there is simply more chaos in the graph. This shows that two principle components accurately capture the motion of the object.

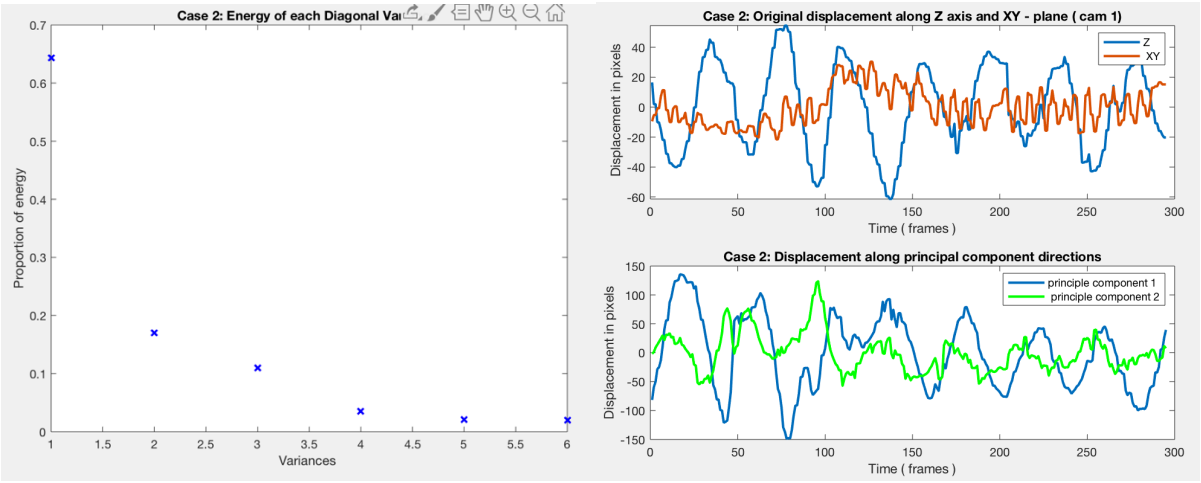

Figure 2 : This shows the principle components and the displacements of the object in two frames in case 2

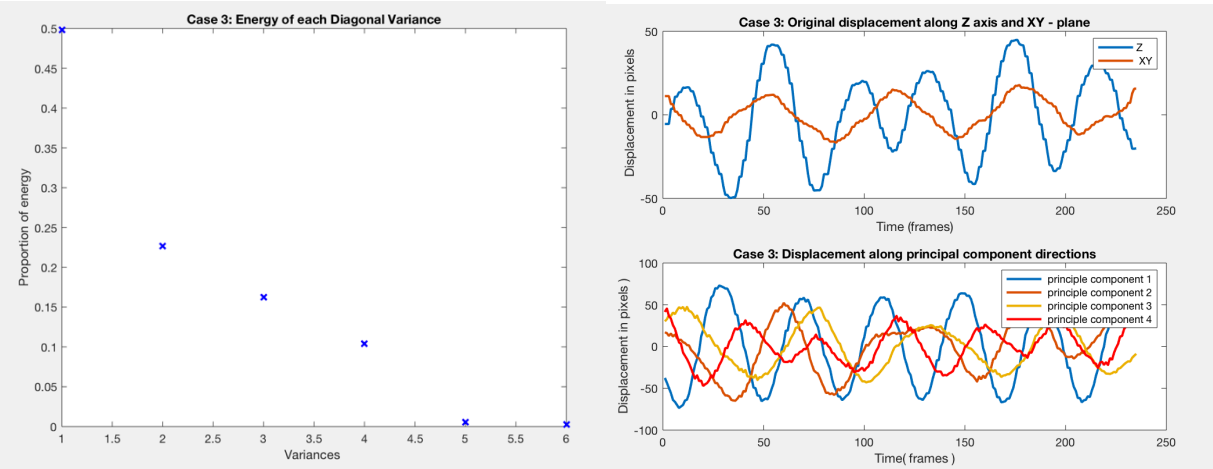

Figure 3 : This shows the principle components and the displacements of the object in two frames in case 3

Case 3:

In the third case, there are 4 principle components, as shown in figure 3. Their values are 0.4985, 0.2267, 0.1627, and 0.1042. Since the initial position of the object has displacement from the z axis. We can see the significant oscillation along the xy plane.

Case 4:

In the fourth case, we have three principle components. Their values are 0.6911, 0.1802, 0.09. In this case, we have both oscillation, rotation, and motion in xy plane. Our PCA has captured the multidimensional nature of the horizontal displacement and rotation, as there is both a pendulum nature, as well as simple harmonic motion.

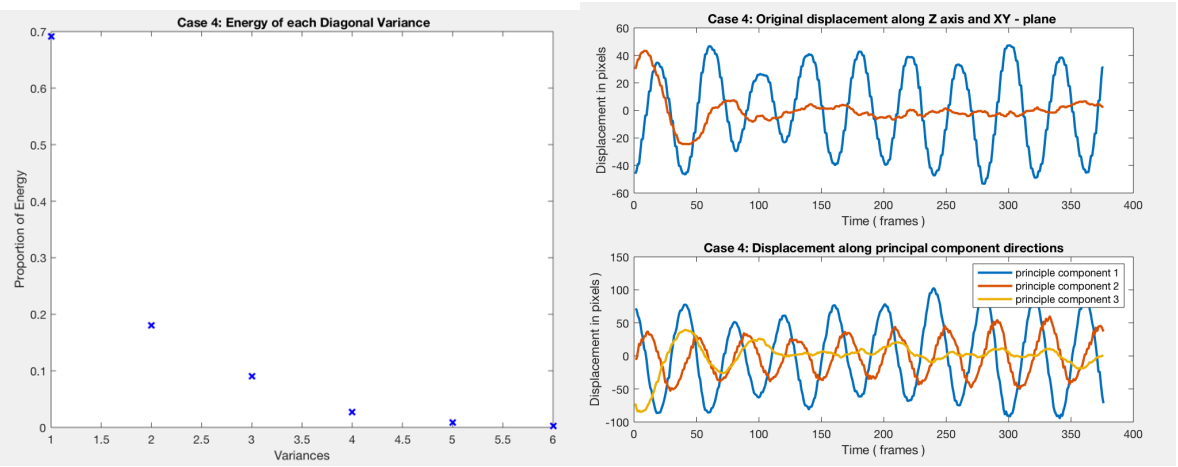

Figure 4 : This shows the principle components and the displacements of the object in two frames in case 4

Summary and Conclusion :

As a summary, principle component algorithm helps us to analyze the motion in the video, without using the governing equation from the physics. We see pretty good results overall. PCA is a power tool to do such analyzation, we can use the principle components to represent the motion in the system, and we are able to eliminate redundant information.

Appendix A (MATLAB functions used) :

Diag() : return a column vector of diagonal values.
Rgb2gray():converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance.
Size() : return the size of a matrix.

Appendix B (MATLAB codes) :

```
%% Load the data (videos) for Test 1/Case 1
clear all; close all; clc

load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')
% implay(vidFrames3_1)

%%
filter = zeros(480 ,640);
filter(190:440, 290:410) = 1;

data1 = load_cropped_data(vidFrames1_1, filter, 240);

%%
filter = zeros(480 ,640);
filter(90:400, 230:360) = 1;

data2 = load_cropped_data(vidFrames2_1, filter, 240);

%%
filter = zeros(480 ,640);
filter(220:340, 250:490) = 1;
```

```
data3 = load_cropped_data(vidFrames3_1, filter, 240);

%%
collected_data = collect(data1, data2, data3);

%% Results for Case 1
[m,n]= size(collected_data);

collected_data = collected_data - repmat(mean(collected_data, 2),1,n);
[U,S,V]= svd(collected_data'/sqrt(n-1));
lambda = diag(S).^2;
Y = collected_data' * V;

figure(1)
plot(1:6, lambda/sum(lambda), 'rx', 'Linewidth', 3);
title("Case 1: Variance and energy");
xlabel("Variances "); ylabel("Proportion of energy");

figure(2)
subplot(2,1,1)
plot(1:216, collected_data(2,:), 1:216, collected_data(1,:), 'Linewidth', 2)
ylabel("Displacement (pixels)"); xlabel("Time(frames)");
title("Case 1: Displacement along Z axis and XY - plane");
legend("Z", "XY")
subplot(2,1,2)
plot(1:216, Y(:,1),'g','Linewidth', 2)
ylabel("Displacement in pixels)"); xlabel("Time (frames)");
title("Case 1: Displacement along principal component directions");
legend("principle component 1")
```

```
%% Load the data for Test 2/Case 2
clear all; close all; clc;

load('cam1_2.mat')
load('cam2_2.mat')
load('cam3_2.mat')

%%
filter = zeros(480 ,640);
filter(200:400, 300:430) = 1;

data1 = load_cropped_data(vidFrames1_2, filter, 240);

%%
filter = zeros(480 ,640);
filter(50:420, 180:440) = 1;

data2 = load_cropped_data(vidFrames2_2, filter, 240);
```

```
%%
filter = zeros(480 ,640);
filter(180:330, 280:470) = 1;

data3 = load_cropped_data(vidFrames3_2, filter, 240);

%%
collected_data = collect(data1, data2, data3);

%% Results for Test 2
[m,n]= size(collected_data);

collected_data = collected_data - repmat(mean(collected_data, 2),1,n);
[U,S,V]= svd(collected_data'/sqrt(n-1));
lambda = diag(S).^2;
Y = collected_data' * V;


figure (3)
plot (1:6,lambda/sum(lambda),'bx','Linewidth', 2) ;
title("Case 2: Energy of each Diagonal Variance ") ;
xlabel("Variances ") ; ylabel("Proportion of energy") ;
figure(4)
subplot(2 ,1 ,1)
plot(1:295, collected_data(2 ,:), 1:295, collected_data(1 ,:) ,'Linewidth',
2)
ylabel(" Displacement in pixels"); xlabel ("Time ( frames )") ;
legend("Z" , " XY ")
title("Case 2: Original displacement along Z axis and XY - plane ( cam 1)") ;
subplot(2,1,2)

plot(1:295 , Y (:,1) ,1:295 , Y(:,2) ,'g','Linewidth', 2)
ylabel("Displacement in pixels") ; xlabel("Time ( frames )") ;
title("Case 2: Displacement along principal component directions") ;
legend("principle component 1 " , " principle component 2")




%% Load the data for Test 3/Case 3
clear all; close all; clc;

load('cam1_3.mat')
load('cam2_3.mat')
load('cam3_3.mat')
%%
filter = zeros(480 ,640);
filter(240:420, 280:380) = 1;

data1 = load_cropped_data(vidFrames1_3, filter, 240);

%%
filter = zeros(480 ,640);
filter(180:380, 240:400) = 1;
```

```
data2 = load_cropped_data(vidFrames2_3, filter, 240);


%%
filter = zeros(480 ,640);
filter(180:330, 240:480) = 1;

data3 = load_cropped_data(vidFrames3_3, filter, 240);

%%
collected_data = collect(data1, data2, data3);

%% Results for Test 3
[m,n]= size(collected_data);

collected_data = collected_data - repmat(mean(collected_data, 2),1,n);
[U,S,V]= svd(collected_data'/sqrt(n-1));
lambda = diag(S).^2;
Y = collected_data' * V;

figure (5)
plot(1:6 , lambda / sum ( lambda ) , 'bx', 'Linewidth', 2) ;
title("Case 3: Energy of each Diagonal Variance") ;
xlabel("Variances ") ; ylabel ("Proportion of energy") ;

figure (6)
subplot (2 ,1 ,1)
plot (1:235 , collected_data(2 ,:), 1:235, collected_data(1,:),'Linewidth',
2)
ylabel (" Displacement in pixels") ; xlabel("Time (frames)") ;
legend("Z" , " XY ")
title (" Case 3: Original displacement along Z axis and XY - plane ") ;
subplot (2 ,1 ,2)
plot (1:235 , Y(: ,1) , 1:235 , Y(: ,2) , 1:235 , Y(: ,3) ,1:235 ,
Y(: ,4) ,'r','Linewidth', 2)
ylabel(" Displacement in pixels ) ") ; xlabel("Time( frames )") ;
title(" Case 3: Displacement along principal component directions ") ;
legend(" principle component 1 " , " principle component 2 " , " principle
component 3 " , " principle component 4 ")




%% Load the data for Test 4/Case 4
clear all; close all; clc;

load('cam1_4.mat')
load('cam2_4.mat')
load('cam3_4.mat')
%%
filter = zeros(480 ,640);
filter(230:440, 330:460) = 1;

data1 = load_cropped_data(vidFrames1_4, filter, 240);
```

```
%%
filter = zeros(480 ,640);
filter(100:360, 230:410) = 1;

data2 = load_cropped_data(vidFrames2_4, filter, 240);


%%
filter = zeros(480 ,640);
filter(140:280, 320:500) = 1;

data3 = load_cropped_data(vidFrames3_4, filter, 230);

%%
[M,I] = min(data1(1:20,2));
data1 = data1(I:end,:);
[M,I] = min(data2(1:20,2));
data2 = data2(I:end,:);
[M,I] = min(data3(1:20,2));
data3 = data3(I:end,:);

data1 = data1(1:length(data3), :);
data2 = data2(1:length(data3), :);

collected_data = [data1'; data2'; data3'];

%% Results for Test 4
[m,n]= size(collected_data);

collected_data = collected_data - repmat(mean(collected_data, 2),1,n);
[U,S,V]= svd(collected_data'/sqrt(n-1));
lambda = diag(S).^2;
Y = collected_data' * V;

figure(7)
plot(1:6 , lambda / sum(lambda) , 'bx', 'Linewidth', 2) ;
title(" Case 4: Energy of each Diagonal Variance") ;
xlabel("Variances") ; ylabel ("Proportion of Energy") ;
figure(8)
subplot(2 ,1 ,1)
plot(1:376 , collected_data(2 ,:) , 1:376, collected_data(1,:),'Linewidth',
2)
ylabel("Displacement in pixels") ; xlabel("Time ( frames )") ;
title("Case 4: Original displacement along Z axis and XY - plane ") ;
subplot(2 ,1 ,2)
plot(1:376 , Y(: ,1) , 1:376 , Y(: ,2) , 1:376 , Y(: ,3) , 'Linewidth', 2)
ylabel("Displacement in pixels )") ; xlabel("Time ( frames )") ;
title("Case 4: Displacement along principal component directions") ;
legend("principle component 1" , "principle component 2" , "principle
component 3")

function data = load_cropped_data(vidFrames, filter, scale)
    numFrames = size(vidFrames,4);
    data = zeros(numFrames, 2);
    for j = 1:numFrames
```

```
        X = vidFrames(:,:,:,j);
        Xg = double(rgb2gray(X));
        X_cropped = Xg .* filter;
        threshold = X_cropped > scale;

        [Y, X] = find(threshold);
        data(j,1) = mean(X);
        data(j,2) = mean(Y);
    end
end

% Collect and arrange the data from three cameras into a single matrix
% Thus the matrix incorporates data points of all measurement types
% taken by each camera over time.
function collected_data = collect(data1, data2, data3)
    [M,I] = min(data1(1:20,2));
    data1 = data1(I:end,:);
    [M,I] = min(data2(1:20,2));
    data2 = data2(I:end,:);
    [M,I] = min(data3(1:20,2));
    data3 = data3(I:end,:);

    % Trim the data to make them a consistent length.
    % For Test1, 2, and 3, the video recorded by camera 1 is (always) the
shortest.
    % Thus trim the other two as the length of video 1.
%     disp(length(data1));
%     disp(length(data2));
%     disp(length(data3));
    data2 = data2(1:length(data1), :);
    data3 = data3(1:length(data1), :);

    collected_data = [data1'; data2'; data3'];
end
```