Homework 3: Principal Component Analysis (PCA)

Yilin Li

February 21 2020

Abstract

This project applies the Singular Value Decomposition (SVD) and Principal Component Analysis (PCA) to extract the position information of the oscillating mass and remove redundant information from 12 videos (created from three different cameras in four different tests). The performance of PCA was tested on increasing complex movements and on noisy measurements and showed by the energy content of each of components.

Sec. I. Introduction and Overview

This project applies Principal Component Analysis (PCA) to videos of an oscillating mass taken by three cameras placed at different locations simultaneously in four different tests (total of 12 videos).

Test 1 is an ideal case, which has a small displacement of the mass in the z direction and the ensuing oscillations. The entire motion is in the z direction with simple harmonic motion. Test 2 is a noisy case, which has camera shakes in it. Test 3 is a case that has horizontal displacement, which means that there is both a pendulum motion and simple harmonic oscillations. Test 4 is a case that has horizontal displacement and rotation. In this case, the mass is released off-center and rotates.

The information of oscillating mass positions were extracted using the flashlight placed on top of the mass and saved as data. After rephased and trimmed data and integrated them into one matrix, the Singular Value Decomposition (SVD) is performed on the new integrated matrix. The project shows and compares the energy content of the components to explores the performance of PCA in different kinds of tests.

The Theoretical Background section introduces the theorems used in the project.

The Algorithm Implementation and Development section lists the algorithms that were implemented in this project and how they were developed.

The Computational Results section shows the plots and analyzes the computational results of MATLAB.

The Summary and Conclusion section summarizes the project and makes a conclusion.

The MATLAB functions and implementations used in this project were attached in Appendix A. The MATLAB codes were attached in Appendix B.

Sec. II. Theoretical Background

The Singular Value Decomposition (SVD)

Let the transformation of a vector x be interpreted as a matrix A geometrically. The transformation can always be decomposed in various orthogonal directions. The Singular Value Decomposition reforms the matrix A into:

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

If A is an n*m matrix, Then the columns of $U \in \mathbb{C}^{m\times m}$ are the left singular vectors of A, the rows of $V \in \mathbb{C}^{n\times n}$ are the right singular vectors of A, and the diagonal elements of $\Sigma \in \mathbb{R}^{n\times m}$ are the singular values of A. The singular values are always non-negative and ordered from largest to smallest. The number of nonzero singular values is the rank (dimension of the range of A) of A. Let $r = rank(A)$. Then, U is a basis for the range of A, and V is a basis for the null space of A. Singular values are the absolute values of the eigenvalues, and singular vectors equals to the eigenvectors.

$$A^T A = V\Lambda V^{-1} \tag{2}$$

The singular values of A are the square roots of the eigenvalues of $A^T A$

If A is a rank-r matrix, then A is the sum of r rank-1 matrices. $A = \sum_{j=1}^{r} \delta_j \overrightarrow{U_j}\overrightarrow{V_j}^T$, $\overrightarrow{U_j}\overrightarrow{V_j}^T$ is the outer product.

For any N so that $0 \leq N \leq r$, we can define a partial sum $A_N = \sum_{j=1}^{N} \delta_j \overrightarrow{U_j}\overrightarrow{V_j}^T$

Then: $||A - A_N||_2 = \delta_{N+1}$

If using the Frobenius norm, then $||A - A_N||_F = \sqrt{\sum_{j=N+1}^{r} \delta_j^2}$

Principal Component Analysis (PCA)

Suppose there is a set of m sensors, and each of them takes n measurements. We can determine how much redundancy exists in two sets of measurements a_i and a_j by computing their covariance

$$\delta_{ij}^2 = \frac{1}{n-1} a_i a_j^T \tag{3}$$

The approach that this project uses to diagonalize the covariance matrix is the SVD. Suppose A has the SVD: $A = U\Sigma V^*$. If we project the data onto the left singular values get $B = U{*}A$, then the covariance matrix of B is:

$$\delta_B^2 = \frac{1}{n-1} BB^T = \frac{1}{n-1} U * U\Sigma V^* (U * U\Sigma V^*)^T = \frac{1}{n-1}\Sigma V^* V \Sigma^T = \frac{1}{n-1}\Sigma^2 \tag{4}$$

Therefore, to determine the directions along which the important dynamics occur, we only need to find the largest diagonal entries of Σ. Actually, if redundancy is present in the measurement, only a few large singular values will be stored in A.

Sec. III. Algorithm Implementation and Development

The algorithm is extremely similar for all four tests.

- Load videos. Find the number of frames for each one by using the *size* command.
- Play every video to decide the width and the location of the filter by visualizing, and create the filter.
- Go through each of the frames of the video in a loop. For each frame, convert it to grayscale using *rgb2gray* and use *double* to store the information in double so we can analyze it.
- Use filter to create a window for tracking movement.
- Find the max values in the grayscale matrix.
- Create a threshold matrix where we found the location that really close to the max intensity (95% of the max value) .
- Use *find* function to locate the indices of these points and use *ind2sub* function to output the coordinates information of all the bright points (the flashlight on the can).
- Average x and y coordinates and store them in the data array.
- Repeat the above steps with the other two videos of this test. At the end, get three matrices.
- Since the videos have different frame numbers and are not in sync, the first frame of each videos has to be changed so it is corresponding with the lowest y coordinate.
- Firstly, find the lowest y coordinate of each data, and use *min* function and *length* function to find the shortest video, and trim the other two matrices so they have the same frame number.
- Add them up to a large matrix, which has 6 rows corresponding to x and y coordinates of the 3 videos.
- Compute mean for each row and subtract it to center the data.
- Perform SVD on the new centered data that is divided by $\sqrt{n-1}$ to get three matrices, [u,s,v].
- Extract the values of the diagonal matrix s and square them to get the eigenvalues, which is the variances for each principal component.
- Plot normalized variances to explore the significant principal components.
- Plot projections of data on the significant principal component orthonormal bases to reconstruct the data.
- Repeat this for other three tests.

Sec. IV. Computational Results

Test 1, Ideal Case

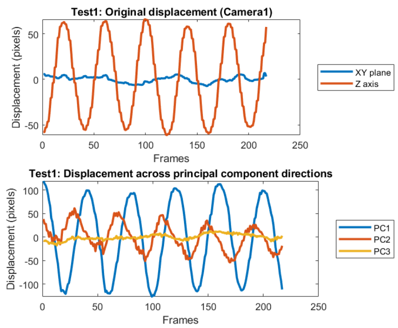

Figure 1: Plots of original displacement and displacement across principal component directions (test 1)

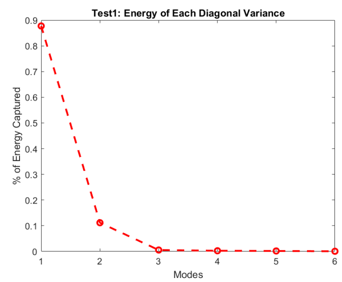

Figure 2: Energy of each Diagonal Variance

The other two cameras have extremely similar plots.

The percentages of the energy captured are: 0.8773 0.1118 0.0056 0.0027 0.0019 0.0007

From the Figure 2, the first principal component corresponds to a 87.7% energy capturing, which is high. The rest of the components had very low energy. The result fits the test; in test 1 the can is only oscillating in one direction, so it should only need one principle component resulted in accurate data. Also, the projection onto the first principal component basis results in very similar data.

Therefore, the ideal case can be accurately reproduced and represented by one principle component, and the PCA gives us good result on this test

Test 2, Noisy case

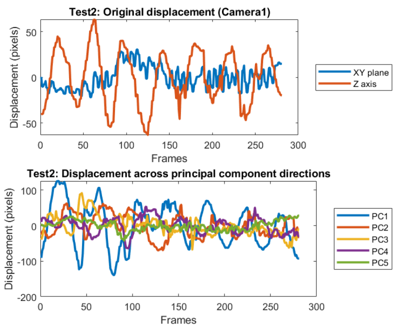

Figure 3: Plots of original displacement and displacement across principal component directions (test 2)

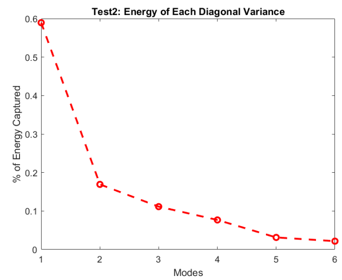

Figure 4: Energy of each Diagonal Variance

The other two cameras have extremely similar plots.

The percentages of the energy captured are: 0.5895 0.1692 0.1114 0.0768 0.0314 0.0217

From the Figure 4, the sum of the first and the second principal components corresponds to a 77% energy capturing, which is pretty low. The rest of the components do not have very low energy. The result fits the test; in test 2 the can is only oscillating in one direction but with noise, so the rest of the components also contain much information. Also, the projection onto the first principal component basis results in very similar data.

Therefore, the noise can throw off some of PCA calculations. However, although there is lots of noise, there is still a clearly oscillatory behavior.

Test 3, Horizontal displacement

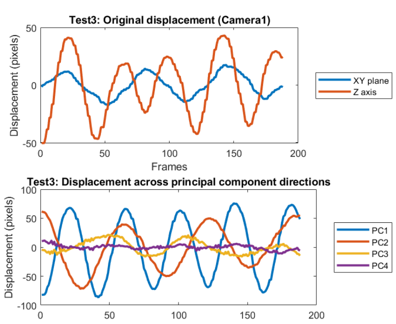

Figure 5: Plots of original displacement and displacement across principal component directions (test 3)

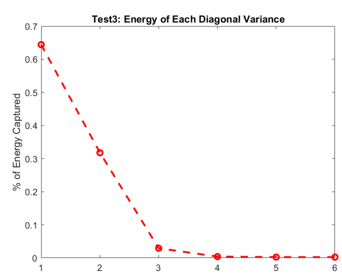

Figure 6: Energy of each Diagonal Variance

The other two cameras have extremely similar plots.

The percentages of the energy captured are: 0.6443 0.3178 0.0292 0.0040 0.0024 0.0023

From the Figure 6, the sum of first three principal components corresponds to a 97% energy capturing, which is really high. The rest of the components had very low energy. The result fits the test; in test 3 the can is producing motion in the xy plane as well as the z direction, so it should need three principle component resulted in accurate data. Also, the projection onto the first principal component basis results in very similar data.

Therefore, the ideal case with the horizontal displacement can be accurately reproduced and represented by three principle component. Additionally, there are large drop off from the first principal component to the third, so PCA gives us good result on this test.

Test 4, Horizontal displacement and rotation

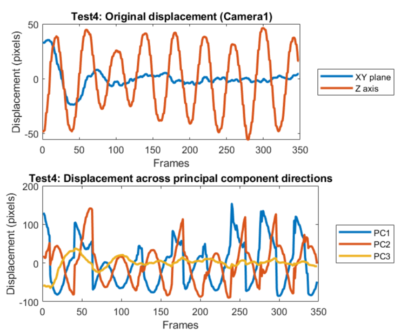

Figure 7: Plots of original displacement and displacement across principal component directions (test 4)

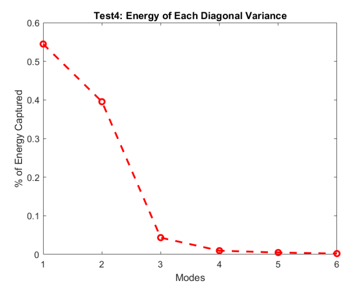

Figure 8: Energy of each Diagonal Variance

The other two cameras have extremely similar plots.

The percentages of the energy captured are: 0.5447 0.3954 0.0434 0.0096 0.0048 0.0021

From the Figure 8, the sum of first three principal components corresponds to a 97% energy capturing, which is really high. The rest of the components had very low energy. The result fits the test; in test 4 the can is is released off-center and rotates so as to produce motion in the xy plane, rotation, and motion in the z direction. Also, the projection onto the first principal component basis results in very similar data.

Therefore, this horizontal displacement and rotation test can be accurately reproduced and represented by three principle component. Additionally, there are large drop off from the second principal component to the third, so PCA captures the multi-dimentional natures and gives us good result on this test.

Sec. V. Summary and Conclusions

This project shows that we are able to track an oscillating mass in different movements with different shooting angles and perform SVD and PCA on the data sets to explore the details of the significant principal components.

To sum up, the PCA method can give us good results in multi-dimentional movements and can handle some noise, but noise is harmful for the PCA method. However, the PCA method is really strong and usefu. It can help us determine the minimum number of principal components that needed to represent a system without knowing the dynamic of the system and reclassify redundant systems into lower order ones.

Appendix A.

MATLAB functions and implementation

- **load(filename)** loads data from filename.
- **sz = size(A)** returns a row vector whose elements are the lengths of the corresponding dimensions of A.
- **implay(filename)** opens the Video Viewer app, displaying the content of the file specified by filename.
- **X = zeros(sz1,...,szN)** returns an sz1-by-...-by-szN array of zeros where sz1,...,szN indicate the size of each dimension.
- **I = rgb2gray(RGB)** converts the truecolor image RGB to the grayscale image I. The rgb2gray function converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance.
- **Y = double(X)** converts the values in X to double precision.
- **M = max(A)** returns the maximum elements of an array.
- **[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.
- **k = find(X)** returns a vector containing the linear indices of each nonzero element in array X.
- **M = mean(A)** returns the mean of the elements of A along the first array dimension whose size does not equal 1.
- **imshow(I)** displays the grayscale image I in a figure.
- **[M,I] = min(A)** returns the index into the operating dimension that corresponds to the minimum value of A for any of the previous syntaxes.
- **L = length(X)** returns the length of the largest array dimension in X.
- **[U,S,V] = svd(A)** performs a singular value decomposition of matrix A, such that A = U*S*V'.
- **D = diag(v)** returns a square diagonal matrix with the elements of vector v on the main diagonal.

- **plot(X,Y)** creates a 2-D line plot of the data in Y versus the corresponding values in X.
- **plot(X1,Y1,...,Xn,Yn)** plots multiple X, Y pairs using the same axes for all lines.

Appendix B. MATLAB codes

Test 1

```
clear; close all; clc;

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

% Find the number of frames
numFrames1_1 = size(vidFrames1_1,4);
numFrames2_1 = size(vidFrames2_1,4);
numFrames3_1 = size(vidFrames3_1,4);

% Play the video1_1 to create the filter
% implay(vidFrames1_1)

% Difine the x,y width of the filter1_1
x_width_11 = 50;
y_width_11 = 130;
% Create the filter for cam1_1
filter1_1 = zeros(480,640);
filter1_1((300-y_width_11):1:(300+y_width_11), (350-x_width_11)
    :1:(350+x_width_11)) = 1;

% Convert videos to grayscale
% Use filter to create a window for tracking movement
% Find the point that has the max intensity
% Save x and y coordinates of that point
data1_1 = [];
for j = 1:numFrames1_1
    C1_1 = vidFrames1_1(:,:,:,j);
    C_to_Gray_11 = rgb2gray(C1_1);
    Gray1_1 = double(C_to_Gray_11);

    Gray1_1f = Gray1_1.*filter1_1;
    white_11 = max(Gray1_1f(:))*0.95;
```

```
    thresh1_1 = Gray1_1f > white_11;
    [Y,X] = ind2sub(size(thresh1_1),find(thresh1_1));

    data1_1 = [data1_1; mean(X),mean(Y)];

    % Plot to check
%     subplot(1,2,1)
%     imshow(uint8((thresh1_1 * max(Gray1_1f(:))))); drawnow
%     title('Thresh');
%     subplot(1,2,2)
%     imshow(uint8(Gray1_1f)); drawnow
%     title('Gray1_1f');
end

close all;


% Play the video2_1 to create the filter
% implay(vidFrames2_1)

% Difine the x,y width of the filter2_1
x_width_21 = 50;
y_width_21 = 150;
% Create the filter for cam2_1
filter2_1 = zeros(480,640);
filter2_1((230-y_width_21):1:(230+y_width_21), (300-x_width_21):1:(300+x_width_21)) = 1;

% Convert videos to grayscale
% Use filter to create a window for tracking movement
% Find the point that has the max intensity
% Save x and y coordinates of that point
data2_1 = [];
for j = 1:numFrames2_1
    C2_1 = vidFrames2_1(:,:,:,j);
    C_to_Gray_21 = rgb2gray(C2_1);
    Gray2_1 = double(C_to_Gray_21);

    Gray2_1f = Gray2_1.*filter2_1;
    white_21 = max(Gray2_1f(:))*0.95;
    thresh2_1 = Gray2_1f > white_21;
    [Y,X] = ind2sub(size(thresh2_1),find(thresh2_1));

    data2_1 = [data2_1; mean(X),mean(Y)];

```

```
    % Plot to check
%     subplot(1,2,1)
%     imshow(uint8((thresh2_1 * max(Gray2_1f(:))))); drawnow
%     title('Thresh');
%     subplot(1,2,2)
%     imshow(uint8(Gray2_1f)); drawnow
%     title('G2_1f');
end

close all;



% Play the video3_1 to create the filter
% implay(vidFrames3_1)

% Difine the x,y width of the filter3_1
x_width_31 = 110;
y_width_31 = 50;
% Create the filter for cam3_1
filter3_1 = zeros(480,640);
filter3_1((290-y_width_31):1:(290+y_width_31), (375-x_width_31):1:(375+x_width_31)) = 1;

% Convert videos to grayscale
% Use filter to create a window for tracking movement
% Find the point that has the max intensity
% Save x and y coordinates of that point
data3_1 = [];
for j = 1:numFrames3_1
    C3_1 = vidFrames3_1(:,:,:,j);
    C_to_Gray_31 = rgb2gray(C3_1);
    Gray3_1 = double(C_to_Gray_31);

    Gray3_1f = Gray3_1.*filter3_1;
    white_31 = max(Gray3_1f(:))*0.95;
    thresh3_1 = Gray3_1f > white_31;
    [Y,X] = ind2sub(size(thresh3_1),find(thresh3_1));

    data3_1 = [data3_1; mean(X),mean(Y)];

    % Plot to check
%     subplot(1,2,1)
%     imshow(uint8((thresh3_1 * max(Gray3_1f(:))))); drawnow
%     title('Thresh');
```

```
%       subplot(1,2,2)
%       imshow(uint8(Gray3_1f)); drawnow
%       title('G3_1f');
end



%%
% Find the lowest Y coodinate of each data
[M,I] = min(data1_1(1:50,2));
data1 = data1_1(I:end,:);

[M,I] = min(data2_1(1:50,2));
data2 = data2_1(I:end,:);

[M,I] = min(data3_1(1:50,2));
data3 = data3_1(I:end,:);

% Find the shortest video
min_length = min([length(data1);length(data2);length(data3)]);
% Trimmed other to make them have the same length
data1_new = data1(1:min_length,:);
data2_new = data2(1:min_length,:);
data3_new = data3(1:min_length,:);

% Add all data into a large one
data_all = [data1_new';data2_new';data3_new'];

% Compute mean for each row
mu = mean(data_all,2);
% Subtract mu
data_all_new = data_all - mu;

% Perform SVD
[u,s,v] = svd(data_all_new'/sqrt(min_length-1));
% Generat eigenvalues
lambda = diag(s).^2;

figure()
plot(lambda/sum(lambda),'ro--','Linewidth',2);
title('Test1: Energy of Each Diagonal Variance')
xlabel('Modes')
ylabel('% of Energy Captured')

figure()
```

```
X = (1:min_length);
subplot(2,1,1)
plot(X,data_all_new(1,:),X,data_all_new(2,:),'Linewidth',2);
title('Test1: Original displacement (Camera1)')
xlabel('Frames')
ylabel('Displacement (pixels)')
legend('XY plane','Z axis','Location','eastoutside')

% subplot(2,2,2)
% plot(X,data_all_new(3,:),X,data_all_new(4,:),'Linewidth',2);
% title('Case1: Original displacement (Camera2)')
% xlabel('Frames')
% ylabel('Displacement (pixels)')
% legend('XY plane','Z axis')
%
%
% subplot(2,2,3)
% plot(X,data_all_new(5,:),X,data_all_new(6,:),'Linewidth',2);
% title('Case1: Original displacement (Camera3)')
% xlabel('Frames')
% ylabel('Displacement (pixels)')
% legend('Z axis','XY plane')


Y = data_all_new' * v;
subplot(2,1,2)
plot(X,Y(:,1),X,Y(:,2),X,Y(:,3),'Linewidth',2);
title('Test1: Displacement across principal component directions')
xlabel('Frames')
ylabel('Displacement (pixels)')
legend('PC1','PC2','PC3','Location','eastoutside')
```

Test 2

```
clear; close all; clc;

% Load videos
load('cam1_2.mat');
load('cam2_2.mat');
load('cam3_2.mat');

% Find the number of frames
numFrames1_2 = size(vidFrames1_2,4);
numFrames2_2 = size(vidFrames2_2,4);
numFrames3_2 = size(vidFrames3_2,4);
```

```

% Play the video1_2 to create the filter
% implay(vidFrames1_2)

% Difine the x,y width of the filter1_2
x_width_12 = 60;
y_width_12 = 110;
% Create the filter for cam1_2
filter1_2 = zeros(480,640);
filter1_2((310-y_width_12):1:(310+y_width_12), (370-x_width_12)
    :1:(370+x_width_12)) = 1;

% Convert videos to grayscale
% Use filter to create a window for tracking movement
% Find the point that has the max intensity
% Save x and y coordinates of that point
data1_2 = [];
for j = 1:numFrames1_2
    C1_2 = vidFrames1_2(:,:,:,j);
    C_to_Gray_12 = rgb2gray(C1_2);
    Gray1_2 = double(C_to_Gray_12);

    Gray1_2f = Gray1_2.*filter1_2;
    white_12 = max(Gray1_2f(:))*0.95;
    thresh1_2 = Gray1_2f > white_12;
    [Y,X] = ind2sub(size(thresh1_2),find(thresh1_2));

    data1_2 = [data1_2; mean(X),mean(Y)];

    % Plot to check
%     subplot(1,2,1)
%     imshow(uint8((thresh1_2 * max(Gray1_2f(:))))); drawnow
%     title('Thresh');
%     subplot(1,2,2)
%     imshow(uint8(Gray1_2f)); drawnow
%     title('Gray1_2f');
end

close all;


% Play the video2_2 to create the filter
% implay(vidFrames2_2)

% Difine the x,y width of the filter2_2
```

```
x_width_22 = 105;
y_width_22 = 170;
% Create the filter for cam2_2
filter2_2 = zeros(480,640);
filter2_2((250-y_width_22):1:(250+y_width_22), (315-x_width_22):1:(315+x_width_22)) = 1;

% Convert videos to grayscale
% Use filter to create a window for tracking movement
% Find the point that has the max intensity
% Save x and y coordinates of that point
data2_2 = [];
for j = 1:numFrames2_2
    C2_2 = vidFrames2_2(:,:,:,j);
    C_to_Gray_22 = rgb2gray(C2_2);
    Gray2_2 = double(C_to_Gray_22);

    Gray2_2f = Gray2_2.*filter2_2;
    white_22 = max(Gray2_2f(:))*0.95;
    thresh2_2 = Gray2_2f > white_22;
    [Y,X] = ind2sub(size(thresh2_2),find(thresh2_2));

    data2_2 = [data2_2; mean(X),mean(Y)];

    % Plot to check
%     subplot(1,2,1)
%     imshow(uint8((thresh2_2 * max(Gray2_2f(:))))); drawnow
%     title('Thresh');
%     subplot(1,2,2)
%     imshow(uint8(Gray2_2f)); drawnow
%     title('G2_2f');
end

close all;



% Play the video3_2 to create the filter
% implay(vidFrames3_2)

% Difine the x,y width of the filter3_2
x_width_32 = 100;
y_width_32 = 70;
% Create the filter for cam3_2
filter3_2 = zeros(480,640);
```

```
filter3_2((270-y_width_32):1:(270+y_width_32), (400-x_width_32)
    :1:(400+x_width_32)) = 1;

% Convert videos to grayscale
% Use filter to create a window for tracking movement
% Find the point that has the max intensity
% Save x and y coordinates of that point
data3_2 = [];
for j = 1:numFrames3_2
    C3_2 = vidFrames3_2(:,:,:,j);
    C_to_Gray_32 = rgb2gray(C3_2);
    Gray3_2 = double(C_to_Gray_32);

    Gray3_2f = Gray3_2.*filter3_2;
    white_32 = max(Gray3_2f(:))*0.95;
    thresh3_2 = Gray3_2f > white_32;
    [Y,X] = ind2sub(size(thresh3_2),find(thresh3_2));

    data3_2 = [data3_2; mean(X),mean(Y)];

    % Plot to check
%     subplot(1,2,1)
%     imshow(uint8((thresh3_2 * max(Gray3_2f(:))))); drawnow
%     title('Thresh');
%     subplot(1,2,2)
%     imshow(uint8(Gray3_2f)); drawnow
%     title('G3_2f');
end



%%
% Find the lowest Y coodinate of each data
[M,I] = min(data1_2(1:50,2));
data1 = data1_2(I:end,:);

[M,I] = min(data2_2(1:50,2));
data2 = data2_2(I:end,:);

[M,I] = min(data3_2(1:50,2));
data3 = data3_2(I:end,:);

% Find the shortest video
min_length = min([length(data1);length(data2);length(data3)]);
% Trimmed other to make them have the same length
```

```
data1_new = data1(1:min_length,:);
data2_new = data2(1:min_length,:);
data3_new = data3(1:min_length,:);

% Add all data into a large one
data_all = [data1_new';data2_new';data3_new'];

% Compute mean for each row
mu = mean(data_all,2);
% Subtract mu
data_all_new = data_all - mu;

% Perform SVD
[u,s,v] = svd(data_all_new'/sqrt(min_length-1));
% Generat eigenvalues
lambda = diag(s).^2;

figure()
plot(lambda/sum(lambda),'ro--','Linewidth',2);
title('Test2: Energy of Each Diagonal Variance')
xlabel('Modes')
ylabel('% of Energy Captured')


figure()
X = (1:min_length);
subplot(2,1,1)
plot(X,data_all_new(1,:),X,data_all_new(2,:),'Linewidth',2);
title('Test2: Original displacement (Camera1)')
xlabel('Frames')
ylabel('Displacement (pixels)')
legend('XY plane','Z axis','Location','eastoutside')

% subplot(2,2,2)
% plot(X,data_all_new(3,:),X,data_all_new(4,:),'Linewidth',2);
% title('Case2: Original displacement (Camera2)')
% xlabel('Frames')
% ylabel('Displacement (pixels)')
% legend('XY plane','Z axis')
%
%
% subplot(2,2,3)
% plot(X,data_all_new(5,:),X,data_all_new(6,:),'Linewidth',2);
% title('Case2: Original displacement (Camera3)')
% xlabel('Frames')
```

```
% ylabel('Displacement (pixels)')
% legend('Z axis','XY plane')


Y = data_all_new' * v;
subplot(2,1,2)
plot(X,Y(:,1),X,Y(:,2),X,Y(:,3),X,Y(:,4),X,Y(:,5),'Linewidth',2);
title('Test2: Displacement across principal component directions')
xlabel('Frames')
ylabel('Displacement (pixels)')
legend('PC1','PC2','PC3','PC4','PC5','Location','eastoutside')
```

Test 3

```
clear; close all; clc;

% Load videos
load('cam1_3.mat');
load('cam2_3.mat');
load('cam3_3.mat');

% Find the number of frames
numFrames1_3 = size(vidFrames1_3,4);
numFrames2_3 = size(vidFrames2_3,4);
numFrames3_3 = size(vidFrames3_3,4);

% Play the video1_3 to create the filter
% implay(vidFrames1_3)

% Difine the x,y width of the filter1_3
x_width_13 = 60;
y_width_13 = 105;
% Create the filter for cam1_3
filter1_3 = zeros(480,640);
filter1_3((315-y_width_13):1:(315+y_width_13), (340-x_width_13)
    :1:(340+x_width_13)) = 1;

% Convert videos to grayscale
% Use filter to create a window for tracking movement
% Find the point that has the max intensity
% Save x and y coordinates of that point
data1_3 = [];
for j = 1:numFrames1_3
    C1_3 = vidFrames1_3(:,:,:,j);
    C_to_Gray_13 = rgb2gray(C1_3);
```

```
    Gray1_3 = double(C_to_Gray_13);

    Gray1_3f = Gray1_3.*filter1_3;
    white_13 = max(Gray1_3f(:))*0.95;
    thresh1_3 = Gray1_3f > white_13;
    [Y,X] = ind2sub(size(thresh1_3),find(thresh1_3));

    data1_3 = [data1_3; mean(X),mean(Y)];

    % Plot to check
%     subplot(1,2,1)
%     imshow(uint8((thresh1_3 * max(Gray1_3f(:))))); drawnow
%     title('Thresh');
%     subplot(1,2,2)
%     imshow(uint8(Gray1_3f)); drawnow
%     title('Gray1_3f');
end

close all;


% Play the video2_3 to create the filter
% implay(vidFrames2_3)

% Difine the x,y width of the filter2_3
x_width_23 = 115;
y_width_23 = 120;
% Create the filter for cam2_3
filter2_3 = zeros(480,640);
filter2_3((290-y_width_23):1:(290+y_width_23), (305-x_width_23)
    :1:(305+x_width_23)) = 1;

% Convert videos to grayscale
% Use filter to create a window for tracking movement
% Find the point that has the max intensity
% Save x and y coordinates of that point
data2_3 = [];
for j = 1:numFrames2_3
    C2_3 = vidFrames2_3(:,:,:,j);
    C_to_Gray_23 = rgb2gray(C2_3);
    Gray2_3 = double(C_to_Gray_23);

    Gray2_3f = Gray2_3.*filter2_3;
    white_23 = max(Gray2_3f(:))*0.95;
    thresh2_3 = Gray2_3f > white_23;
```

```
    [Y,X] = ind2sub(size(thresh2_3),find(thresh2_3));

    data2_3 = [data2_3; mean(X),mean(Y)];

    % Plot to check
%     subplot(1,2,1)
%     imshow(uint8((thresh2_3 * max(Gray2_3f(:))))); drawnow
%     title('Thresh');
%     subplot(1,2,2)
%     imshow(uint8(Gray2_3f)); drawnow
%     title('G2_3f');
end

close all;



% Play the video3_3 to create the filter
% implay(vidFrames3_3)

% Difine the x,y width of the filter3_3
x_width_33 = 100;
y_width_33 = 135;
% Create the filter for cam3_3
filter3_3 = zeros(480,640);
filter3_3((205-y_width_33):1:(205+y_width_33), (370-x_width_33)
    :1:(370+x_width_33)) = 1;

% Convert videos to grayscale
% Use filter to create a window for tracking movement
% Find the point that has the max intensity
% Save x and y coordinates of that point
data3_3 = [];
for j = 1:numFrames3_3
    C3_3 = vidFrames3_3(:,:,:,j);
    C_to_Gray_33 = rgb2gray(C3_3);
    Gray3_3 = double(C_to_Gray_33);

    Gray3_3f = Gray3_3.*filter3_3;
    white_33 = max(Gray3_3f(:))*0.95;
    thresh3_3 = Gray3_3f > white_33;
    [Y,X] = ind2sub(size(thresh3_3),find(thresh3_3));

    data3_3 = [data3_3; mean(X),mean(Y)];

```

```
    % Plot to check
%     subplot(1,2,1)
%     imshow(uint8((thresh3_3 * max(Gray3_3f(:))))); drawnow
%     title('Thresh');
%     subplot(1,2,2)
%     imshow(uint8(Gray3_3f)); drawnow
%     title('G3_3f');
end



%%
% Find the lowest Y coodinate of each data
[M,I] = min(data1_3(1:50,2));
data1 = data1_3(I:end,:);

[M,I] = min(data2_3(1:50,2));
data2 = data2_3(I:end,:);

[M,I] = min(data3_3(1:50,2));
data3 = data3_3(I:end,:);

% Find the shortest video
min_length = min([length(data1);length(data2);length(data3)]);
% Trimmed other to make them have the same length
data1_new = data1(1:min_length,:);
data2_new = data2(1:min_length,:);
data3_new = data3(1:min_length,:);

% Add all data into a large one
data_all = [data1_new';data2_new';data3_new'];

% Compute mean for each row
mu = mean(data_all,2);
% Subtract mu
data_all_new = data_all - mu;

% Perform SVD
[u,s,v] = svd(data_all_new'/sqrt(min_length-1));
% Generat eigenvalues
lambda = diag(s).^2;

figure()
plot(lambda/sum(lambda),'ro--','Linewidth',2);
title('Test3: Energy of Each Diagonal Variance')
```

```
xlabel('Modes')
ylabel('% of Energy Captured')

figure()
X = (1:min_length);
subplot(2,1,1)
plot(X,data_all_new(1,:),X,data_all_new(2,:),'Linewidth',2);
title('Test3: Original displacement (Camera1)')
xlabel('Frames')
ylabel('Displacement (pixels)')
legend('XY plane','Z axis','Location','eastoutside')

% subplot(2,2,2)
% plot(X,data_all_new(3,:),X,data_all_new(4,:),'Linewidth',2);
% title('Case2: Original displacement (Camera2)')
% xlabel('Frames')
% ylabel('Displacement (pixels)')
% legend('XY plane','Z axis')
%
%
% subplot(2,2,3)
% plot(X,data_all_new(5,:),X,data_all_new(6,:),'Linewidth',2);
% title('Case2: Original displacement (Camera3)')
% xlabel('Frames')
% ylabel('Displacement (pixels)')
% legend('Z axis','XY plane')


Y = data_all_new' * v;
subplot(2,1,2)
plot(X,Y(:,1),X,Y(:,2),X,Y(:,3),X,Y(:,4),'Linewidth',2);
title('Test3: Displacement across principal component directions')
xlabel('Frames')
ylabel('Displacement (pixels)')
legend('PC1','PC2','PC3','PC4','Location','eastoutside')
```

Test 4

```
clear; close all; clc;

% Load videos
load('cam1_4.mat');
load('cam2_4.mat');
load('cam3_4.mat');

```

```
% Find the number of frames
numFrames1_4 = size(vidFrames1_4,4);
numFrames2_4 = size(vidFrames2_4,4);
numFrames3_4 = size(vidFrames3_4,4);

% Play the video1_3 to create the filter
% implay(vidFrames1_4)

% Difine the x,y width of the filter1_3
x_width_14 = 50;
y_width_14 = 130;
% Create the filter for cam1_4
filter1_4 = zeros(480,640);
filter1_4((300-y_width_14):1:(300+y_width_14), (350-x_width_14)
    :1:(350+x_width_14)) = 1;

% Convert videos to grayscale
% Use filter to create a window for tracking movement
% Find the point that has the max intensity
% Save x and y coordinates of that point
data1_4 = [];
for j = 1:numFrames1_4
    C1_4 = vidFrames1_4(:,:,:,j);
    C_to_Gray_14 = rgb2gray(C1_4);
    Gray1_4 = double(C_to_Gray_14);

    Gray1_4f = Gray1_4.*filter1_4;
    white_14 = max(Gray1_4f(:))*0.95;
    thresh1_4 = Gray1_4f > white_14;
    [Y,X] = ind2sub(size(thresh1_4),find(thresh1_4));

    data1_4 = [data1_4; mean(X),mean(Y)];

    % Plot to check
%     subplot(1,2,1)
%     imshow(uint8((thresh1_4 * max(Gray1_4f(:))))); drawnow
%     title('Thresh');
%     subplot(1,2,2)
%     imshow(uint8(Gray1_4f)); drawnow
%     title('Gray1_4f');
end

close all;


```

```
% Play the video2_4 to create the filter
% implay(vidFrames2_4)

% Difine the x,y width of the filter2_4
x_width_24 = 50;
y_width_24 = 150;
% Create the filter for cam2_4
filter2_4 = zeros(480,640);
filter2_4((230-y_width_24):1:(230+y_width_24), (300-x_width_24):1:(300+x_width_24)) = 1;

% Convert videos to grayscale
% Use filter to create a window for tracking movement
% Find the point that has the max intensity
% Save x and y coordinates of that point
data2_4 = [];
for j = 1:numFrames2_4
    C2_4 = vidFrames2_4(:,:,:,j);
    C_to_Gray_24 = rgb2gray(C2_4);
    Gray2_4 = double(C_to_Gray_24);

    Gray2_4f = Gray2_4.*filter2_4;
    white_24 = max(Gray2_4f(:))*0.95;
    thresh2_4 = Gray2_4f > white_24;
    [Y,X] = ind2sub(size(thresh2_4),find(thresh2_4));

    data2_4 = [data2_4; mean(X),mean(Y)];

    % Plot to check
%     subplot(1,2,1)
%     imshow(uint8((thresh2_4 * max(Gray2_4f(:))))); drawnow
%     title('Thresh');
%     subplot(1,2,2)
%     imshow(uint8(Gray2_4f)); drawnow
%     title('G2_4f');
end

close all;



% Play the video3_4 to create the filter
% implay(vidFrames3_4)

% Difine the x,y width of the filter3_4
```

```
x_width_34 = 110;
y_width_34 = 50;
% Create the filter for cam3_4
filter3_4 = zeros(480,640);
filter3_4((290-y_width_34):1:(290+y_width_34), (375-x_width_34)
    :1:(375+x_width_34)) = 1;

% Convert videos to grayscale
% Use filter to create a window for tracking movement
% Find the point that has the max intensity
% Save x and y coordinates of that point
data3_4 = [];
for j = 1:numFrames3_4
    C3_4 = vidFrames3_4(:,:,:,j);
    C_to_Gray_34 = rgb2gray(C3_4);
    Gray3_4 = double(C_to_Gray_34);

    Gray3_4f = Gray3_4.*filter3_4;
    white_34 = max(Gray3_4f(:))*0.95;
    thresh3_4 = Gray3_4f > white_34;
    [Y,X] = ind2sub(size(thresh3_4),find(thresh3_4));

    data3_4 = [data3_4; mean(X),mean(Y)];

    % Plot to check
%     subplot(1,2,1)
%     imshow(uint8((thresh3_4 * max(Gray3_4f(:))))); drawnow
%     title('Thresh');
%     subplot(1,2,2)
%     imshow(uint8(Gray3_4f)); drawnow
%     title('G3_4f');
end



%%
% Find the lowest Y coodinate of each data
[M,I] = min(data1_4(1:50,2));
data1 = data1_4(I:end,:);

[M,I] = min(data2_4(1:50,2));
data2 = data2_4(I:end,:);

[M,I] = min(data3_4(1:50,2));
data3 = data3_4(I:end,:);
```

```
% Find the shortest video
min_length = min([length(data1);length(data2);length(data3)]);
% Trimmed other to make them have the same length
data1_new = data1(1:min_length,:);
data2_new = data2(1:min_length,:);
data3_new = data3(1:min_length,:);

% Add all data into a large one
data_all = [data1_new';data2_new';data3_new'];

% Compute mean for each row
mu = mean(data_all,2);
% Subtract mu
data_all_new = data_all - mu;

% Perform SVD
[u,s,v] = svd(data_all_new'/sqrt(min_length-1));
% Generat eigenvalues
lambda = diag(s).^2;

figure()
plot(lambda/sum(lambda),'ro--','Linewidth',2);
title('Test4: Energy of Each Diagonal Variance')
xlabel('Modes')
ylabel('% of Energy Captured')


figure()
X = (1:min_length);
subplot(2,1,1)
plot(X,data_all_new(1,:),X,data_all_new(2,:),'Linewidth',2);
title('Test4: Original displacement (Camera1)')
xlabel('Frames')
ylabel('Displacement (pixels)')
legend('XY plane','Z axis','Location','eastoutside')

% subplot(2,2,2)
% plot(X,data_all_new(3,:),X,data_all_new(4,:),'Linewidth',2);
% title('Case2: Original displacement (Camera2)')
% xlabel('Frames')
% ylabel('Displacement (pixels)')
% legend('XY plane','Z axis')
%
%
```

```
% subplot(2,2,3)
% plot(X,data_all_new(5,:),X,data_all_new(6,:),'Linewidth',2);
% title('Case2: Original displacement (Camera3)')
% xlabel('Frames')
% ylabel('Displacement (pixels)')
% legend('Z axis','XY plane')


Y = data_all_new' * v;
subplot(2,1,2)
plot(X,Y(:,1),X,Y(:,2),X,Y(:,3),'Linewidth',2);
title('Test4: Displacement across principal component directions')
xlabel('Frames')
ylabel('Displacement (pixels)')
legend('PC1','PC2','PC3','Location','eastoutside')
```