AMATH 582 Homework 3

Reed Nomura

February 21, 2020

Abstract

This paper explores the use of principal component analysis in order to evaluate movement within footage. Specifically we will be analyzing four tests of a paint can suspended on a spring. Each test is filmed by three different angles. The paint can will be tracked in each shot and then we will use singular value decomposition to clarify what we are seeing in the data collected.

1 Introduction and Overview

Principal component analysis is a procedure used to transform data into orthogonal dimensions in which our data exists. These dimensions provide important insight into the coordinate system in which our data exists and is best interpreted. In this paper we will be using principal component analysis to understand the movement of a paint can suspended by a spring. We will be examining four different tests. The first test consists of the can simply bouncing up and down on the spring. The second test is similar to the first with some added noise by shaking the cameras as the experiment is being filmed. The first test adds another dimension by swinging the paint can side to side while it is also bouncing up and down. Finally, the fourth test involves having the paint can spin while simultaneously bouncing up and down and swinging side to side. Each of these tests are filmed by three different angles to provide differing points of observation.

2 Theoretical Background

2.1 Singular Value Decomposition (SVD)

A singular value decomposition (SVD) is a factorization of a matrix into a number of constitutive components all of which have a specific meaning in applications.[1]. SVD dcomposition takes on the form

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

with

$$U \in \mathbb{C}^{mxm} \text{ is unitary} \tag{2}$$

$$V \in \mathbb{C}^{nxn} \text{ is unitary} \tag{3}$$

$$\Sigma \in \mathbb{R}^{mxn} \text{ is diagonal} \tag{4}$$

2.2 Eigenvector Decomposition

The most straightforward way to diagonalize the covariance matrix is by making the observation that XXT is a square, symmetric m × m matrix, i.e. it is self-adjoint so that the m eigenvalues are real and distinct. [1] Linear algebra provides theorems which state that such a matrix can be rewritten as

$$XX^T = S\Lambda S^{-1} = S\Lambda S^T \tag{5}$$

Instead of working with X, we are able to work with

$$Y = S^T X \tag{6}$$

It then follows that

$$C_Y = \frac{1}{n-1} YY^T \tag{7}$$

$$= \frac{1}{n-1}(S^T X)(S^T X)^T \tag{8}$$

$$= \frac{1}{n-1}(S^T X)(X^T S) \tag{9}$$

$$= \frac{1}{n-1} S^T XX^T S \tag{10}$$

$$= \frac{1}{n-1} S^T \Lambda S^T S \tag{11}$$

Where

$$\Lambda = \begin{bmatrix} \Lambda_1 & & & \\ & \Lambda_2 & & \\ & & \ddots & \\ & & & \Lambda_n \end{bmatrix} \tag{12}$$

Hence

$$C_Y = \frac{1}{n-1}\Lambda \tag{13}$$

In this basis, the principal components are the eigenvectors of XXT with the interpretation that the jth diagonal value of CY is the variance of X along xj, the jth column of S [1].

2.3 SVD for Diagonalizing Covariance Matrix

A second method for diagonalizing the co- variance matrix is the SVD method. In this case, the SVD can diagonalize any matrix by working in the appropriate pair of bases U and V as outlined in the first lecture of this section. Thus by defining the transformed variable

$$Y = U^* X \tag{14}$$

where U is the unitary transformation associated with the SVD: $X = U\ \Sigma\ V^*$. Just as in the eigenvalue/eigenvector formulation, we then compute the variance in Y:

$$C_Y = \frac{1}{n-1} Y Y^T \tag{15}$$

$$= \frac{1}{n-1} (U^* X)(U^* X)^T \tag{16}$$

$$= \frac{1}{n-1} U^* (X X^T) U \tag{17}$$

$$= \frac{1}{n-1} U^* U \Sigma^2 U U^* \tag{18}$$

$$= \frac{1}{n-1} \Sigma^2 \tag{19}$$

This makes explicit the connection between the SVD and the eigenvalue method, namely that $\Sigma^2 = \Lambda$[1].

3 Algorithm Implementation and Development

3.1 Outlines

1. Load Footage
2. Standardize footage
3. Track Paint Can
4. Standardize data and Put into Matrix
5. Perform SVD
6. View Data

3.2 Algorithms

Algorithm 1: Analyze footage using SVD

```
Import Footage
Standardize pixel dimensions and frame count
Set Window Width
for i = 1 : 3 do
    Track Paint Can for Camera i
    Standardize pixel locations
end for
Store Pixel location vectors in a Matrix
Perform SVD on Matrix
Plot Sigma Values
Plot appropriate modes
```

Algorithm 2: Track Paint Can

```
for i = 1 : Number of Frames do
    Convert frame i to grayscale
    Find the pixel location of the brightest pixel
    Store each x and y value in a respective vector
end for
```

4 Computational Results

4.1 Test 1 (Ideal Case)

In this test each of the three clips were trimmed to 200 x 200 pixel square shots in order to try and avoid unnecessary background information interfering with the data. In order to make the clips uniform, each clip was also trimmed to the length of the shortest clip (226 frames). The flashlight on top of the paint can was used as a tracking point as the can was bounced up and down. The graph below shows the movement of the paint can in both the x and y axes.

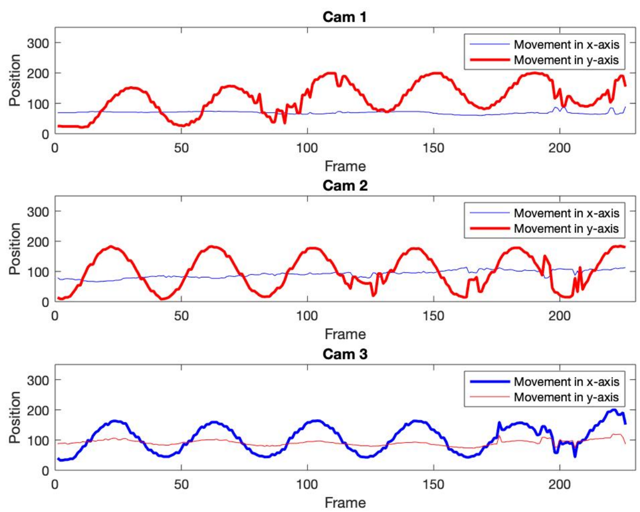


Figure 1: Initial Tracking Plots for Test 1

In this graph we can see the periodic motion of the paint can. For the first two camera angles, this motion is on the y axis. However, for the third camera angle, the can is moving on the x-axis. These camera orientations will be consistent for all four tests. The x and y locations at each frame were stored in vectors. The vectors were then combined to form a $6x226$ matrix. I then performed the singular value decomposition on the matrix.

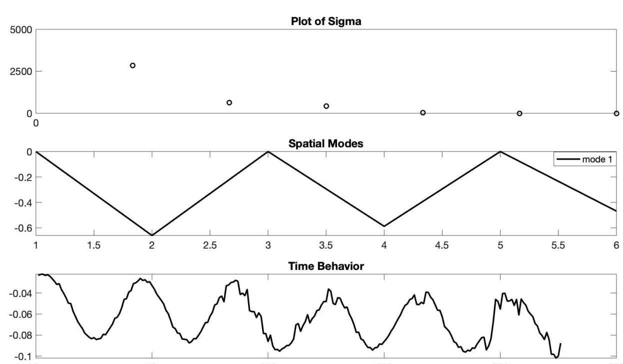


Figure 2: Singular Value Decomposition for Test 1

The first plot in the figure above is all of the sigma values for the svd. The relative size of σ_1 suggests that this is a rank 1 set of data. However, $sigma_2$ and $sigma_3$ are high enough that they are not rounded off to zero. This is understandable since we could expect some minor correlated movement in the x and z axes in all three shots. Since we know that this is a rank one test, we are going to ignore movement in the other modes. The second plot in figure 2 shows the graph of the values of the first column of U and the third figure is the graph of the first column of V.

4.2 Test 2 (Noisy Case)

This test is very similar to the first. However, there is added noise created by shaking the camera for the duration of the test. Each of the three clips were trimmed to 250 x 250 pixel square shots. These squares needed to be bigger than the first experiment to compensate for the shaking of the cameras. Each clip was also trimmed to the length of the shortest clip (314 frames). Just like the first test, the flashlight was tracked and the graph below shows the movement of the paint can in both the x and y axes.

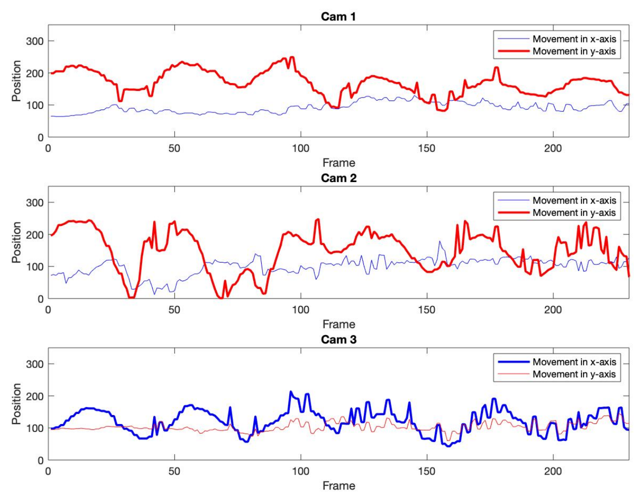


Figure 3: Initial Tracking Plots for Test 2

In this graph we can still partially make out the periodic motion of the paint can. However, the noise has now made it harder to see the movement as clearly as we did in figure 1. Similar to test one we create a matrix of all of the x and y coordinates. This time I made a $6x314$ matrix as there are 314 frames to account for. I then performed the singular value decomposition on the matrix.

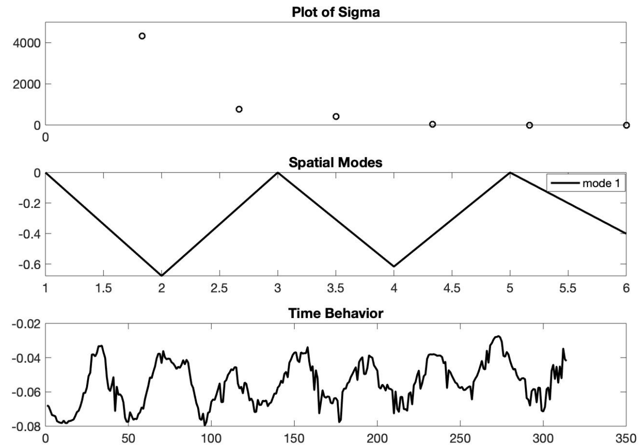


Figure 4: Singular Value Decomposition for Test 2

Once again we see that the relative size of σ_1 suggests that this is a rank 1 set of data. As we expect, the second and third plots look very similar to the ones in figure 4.

4.3 Test 3 (Horizontal Displacement)

In this test there is movement added in the x axis by swinging the object as it is bouncing. Each clips was trimmed to 200 x 200 squares 237 frames long. Just like the first test, the flashlight was tracked and the graph below shows the movement of the paint can in both the x and y axes.

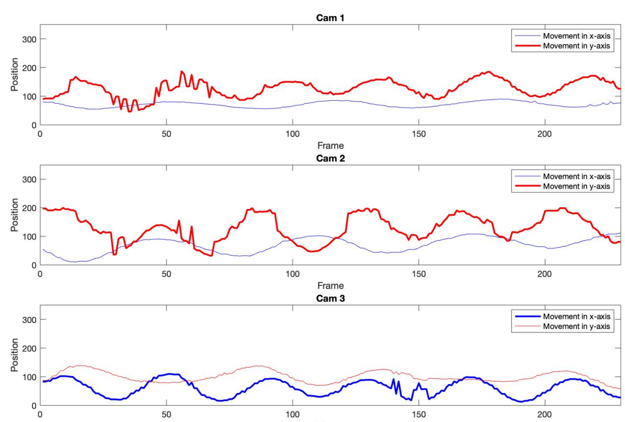


Figure 5: Initial Tracking Plots for Test 3

In this graph we can make out the periodic motion of the paint can bouncing. Additionally, we can see movement in a second axis as well. We create a matrix of all of the x and y coordinates. This time I made a $6x314$ matrix as there are 314 frames to account for. I then performed the singular value decomposition on the matrix.

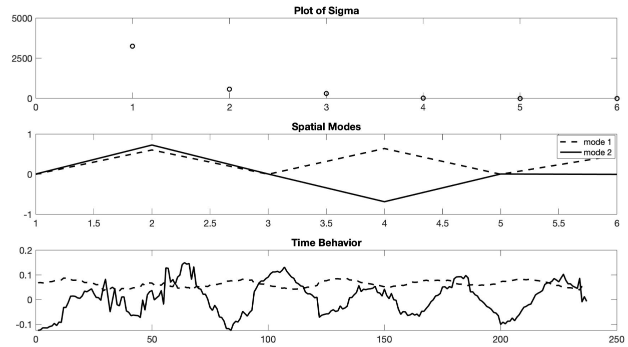


Figure 6: Singular Value Decomposition for Test 3

Due to the size of σ_1 and σ_2, we can conclude that this is a rank 2 test. We graph the the two modes.

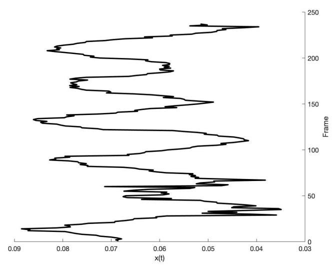
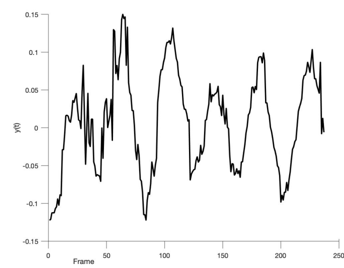
Change in X	Change in Y
	

Table 1: Graph of Test 3 change in time

In the figure above we can see the way that the paint can changes on the x axis with respect to time on the right and the y axis on the right.

4.4 Test 4

In this test there is movement added in the z axis by rotating the object as it is bouncing and swinging side-to-side. Each clips was trimmed to 200 x 200 squares 392 frames long. Just like the tests before it, the flashlight was tracked and the graph below shows the movement of the paint can in both the x and y axes. However, there are times in the first clip that the flashlight is turned away from the camera. In order to compensate for that, the pink of the flashlight was turned bright white and tracked as well.

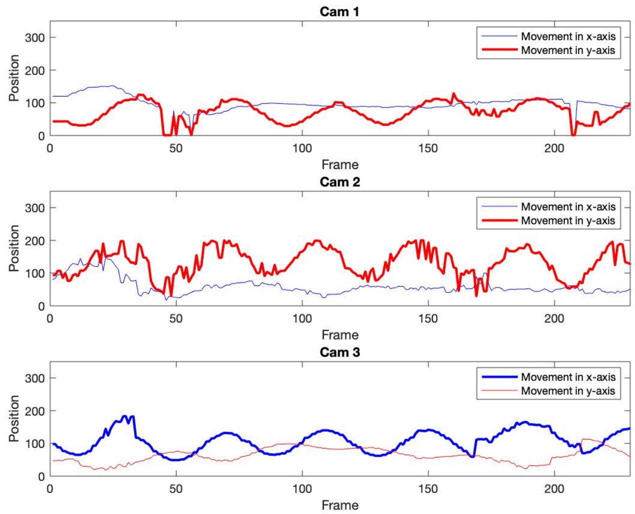


Figure 7: Initial Tracking Plots for Test 4

In this graph we can make out the periodic motion of the paint can bouncing. However, the it has become harder to identify the movement in the other axis. We create a matrix of all of the x and y coordinates. This time I made a $6x392$ matrix as there are 392 frames to account for. I then performed the singular value decomposition on the matrix.

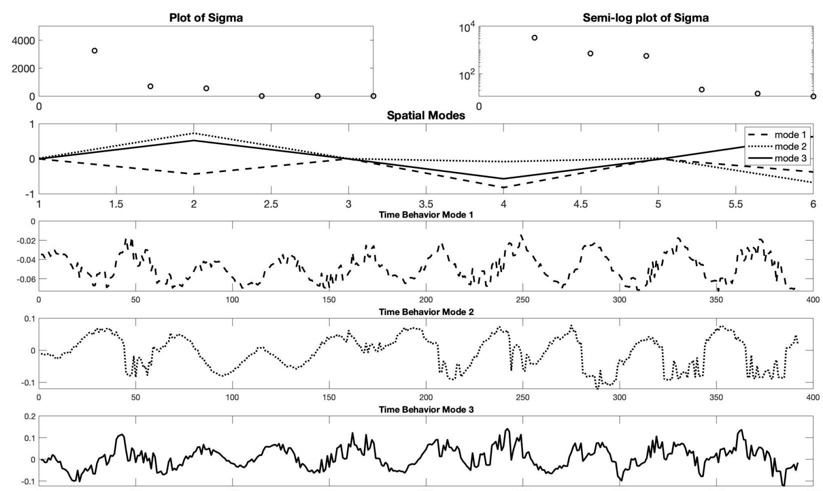


Figure 8: Singular Value Decomposition for Test 4

Due to the size of σ_1, σ_2, and σ_3 we can conclude that this is a rank 3 test. We can see the graph the the behavior of the three modes with respect to time in the three figures at the bottom of the figure 8. This follows with our knowledge of the test. Since the can was bouncing, swinging, and spinning, we expected to see three dimensions all engaged.

5 Summary and Conclusions

Each test, although slightly different, essentially included the same basics steps. In each case we resized the images and frame count. We then tracked the can's movement through the frames. After creating a matrix of all of the recorded x and y positions, we performed the svd on the matrix. The plot of the sigma values clued us in to the rank of the test. We were then able to graph the appropriate number of modes in order to see the movement along each orthogonal dimension that the svd discovered.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

[2] *MathWorks Website*. URL: `https://www.mathworks.com/help/matlab/index.html`.

Appendix A MATLAB Functions

- `sz = size(A)`: returns a row vector whose elements are the lengths of the corresponding dimensions of A. For example, if A is a 3-by-4 matrix, then size(A) returns the vector [3 4]. [2]
- `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. [2]
- `[M,I] = max()`: returns the index into the operating dimension that corresponds to the maximum value of A for any of the previous syntaxes. [2]
- `[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. Here sz is a vector with two elements, where sz(1) specifies the number of rows and sz(2) specifies the number of columns.[2]
- `A(:)` :reshapes all elements of A into a single column vector. This has no effect if A is already a column vector.
- `size(A)`: returns a row vector whose elements are the lengths of the corresponding dimensions of A. For example, if A is a 3-by-4 matrix, then size(A) returns the vector [3 4].[2]
- `B = repmat(A,n)`: returns an array containing n copies of A in the row and column dimensions. The size of B is size(A)*n when A is a matrix [2].
- `sigma = svd(A)`: returns a vector sigma containing the singular values of a symbolic matrix A. [2]
- `y = linspace(x1,x2,n)`: generates n points. The spacing between the points is $\frac{x2-x1}{n-1}$ [2]
- `D = diag(v)` : returns a square diagonal matrix with the elements of vector v on the main diagonal. [2].

Appendix B MATLAB Code

This code can be found at: https://github.com/ReedNomura/AMATH-582/blob/master/Homework4.m

```
 %AMATH 582 Homework 3

%% Ideal Case
clear all; close all; clc; %Start Fresh
load('cam1_1.mat');
load('cam2_1.mat');
```

```
load('cam3_1.mat');

%% Initial Dimension Check
[A1_1,B1_1,C1_1,D1_1] = size(vidFrames1_1);
[A2_1,B2_1,C2_1,D2_1] = size(vidFrames2_1);
[A3_1,B3_1,C3_1,D3_1] = size(vidFrames3_1);
%% Resizing videos to 200 by 200 and 226 Frames
vidFrames1_1 = vidFrames1_1(201:400,251:450,:,:);
vidFrames2_1 = vidFrames2_1(101:300,201:400,:,18:243); % This video starts approximately ...
    18 frames earlier than Camera 1 and ends later as well
vidFrames3_1 = vidFrames3_1(176:375,251:450,:,7:232); % This video starts approximately 7
[A1_1,B1_1,C1_1,D1_1] = size(vidFrames1_1);
[A2_1,B2_1,C2_1,D2_1] = size(vidFrames2_1);
[A3_1,B3_1,C3_1,D3_1] = size(vidFrames3_1);
X1_1 = [];
Y1_1 = [];
X2_1 = [];
Y2_1 = [];
X3_1 = [];
Y3_1 = [];
%% Camera 1
for i = 1:D1_1
    img = rgb2gray(vidFrames1_1(:,:,:,i)); %Make every frame Black and White
    [Max, Ind] = max(img(:)); %Reshapes img to vector and find the max value of img and ...
        where it is indexed
    [y1_1, x1_1] = ind2sub(size(img),Ind);
    X1_1 = [X1_1, x1_1];
    Y1_1 = [Y1_1, y1_1];
end
%% Camera 2
for i = 1:D2_1
    img = rgb2gray(vidFrames2_1(:,:,:,i));
    [Max, Ind] = max(img(:));
    [y2_1, x2_1] = ind2sub(size(img),Ind);
    X2_1 = [X2_1, x2_1];
    Y2_1 = [Y2_1, y2_1];
end
%% Camera 3
for i = 1:D3_1
    img = rgb2gray(vidFrames3_1(:,:,:,i));
    [Max, Ind] = max(img(:));
    [y3_1, x3_1] = ind2sub(size(img),Ind);
    X3_1 = [X3_1, x3_1];
    Y3_1 = [Y3_1, y3_1];
end
%% Plots for Ideal Case
figure()
subplot(3,1,1)
plot(X1_1,'b')
hold on;
plot(Y1_1, 'r', 'Linewidth',[2])
hold off;
xlabel('Frame')
xlim([0,230])
ylabel('Position')
ylim([0,350])
legend('Movement in x-axis','Movement in y-axis')
title('Cam 1')

subplot(3,1,2)
plot(X2_1,'b')
hold on;
plot(Y2_1, 'r', 'Linewidth',[2])
hold off;
xlabel('Frame')
xlim([0,230])
ylabel('Position')
ylim([0,350])
```

```
legend('Movement in x-axis','Movement in y-axis')
title('Cam 2')

subplot(3,1,3)
plot(X3_1,'b', 'Linewidth',[2])
hold on;
plot(Y3_1, 'r')
hold off;
xlabel('Frame')
xlim([0,230])
ylabel('Position')
ylim([0,350])
legend('Movement in x-axis','Movement in y-axis')
title('Cam 3')

%% Standardize
X1_1 = zscore(X1_1);
X2_1 = zscore(X2_1);
X3_1 = zscore(X3_1);
Y1_1 = zscore(Y1_1);
Y2_1 = zscore(Y2_1);
Y3_1 = zscore(Y3_1);

%% SVD
X1 = [X1_1;Y1_1;X2_1;Y2_1;X3_1;Y3_1];
x = linspace(1,6,6);
t = linspace(1,226,226);
[m,n]=size(X1); % compute data size
mn = mean(X1,2); % compute mean for each row
X1 = X1-repmat(mn,1,n); % subtract mean
[u,s,v]=svd(X1'/sqrt(n-1)); % perform the SVD
sig = diag(s).^2; % produce diagonal variances
Yu = u*X1'; % produce the principal components projection
Yv = X1'*v;

%%
figure()
sig=diag(s);

subplot(3,1,1), plot(sig,'ko','Linewidth',[1.5])
axis([0 6 0 2000])
set(gca,'Fontsize',[13],'Ytick', [0 1000 2000], 'Xtick',[0  1 2 3 4 5 6])
text(20,40,'(a)','Fontsize',[13])
title('Plot of Sigma')

% subplot(3,2,2), semilogy(sig,'ko','Linewidth',[1.5])
%  axis([0 6 0 10000])
%  set(gca,'Fontsize',[13],'Ytick',[0, 10^0,10^2, 10^4], 'Xtick',[0  10 15 20 25]);
%  text(20,40,'(b)','Fontsize',[13])
%  title('Semi-log plot of Sigma')


subplot(3,1,2) % spatial modes
plot(x,u(:,1),'k','Linewidth',[2])
set(gca,'Fontsize',[13])
legend('mode 1','Location','NorthWest')
title('Spatial Modes')

subplot(3,1,3) % time behavior
plot(t,v(:,1),'k','Linewidth',[2])
set(gca,'Fontsize',[13])
title('Time Behavior')
%% Movie
figure()
for k = 1:D1_1
    ColorScale = (D1_1 - k)/D1_1;
    CSM = [ColorScale, ColorScale, ColorScale];
    Frame = k;
```

```
    plot(0, v(k,1),'.', 'MarkerSize', [10], 'color', [0 0 0])
    axis([-.005 .005 .025 .125])
    set(gca,'Fontsize',[13],'Ytick',[],'Xtick',[])
    xlabel('Axis 1')
    ylabel('Axis 2')
    title('Movement in Relevant Axes')

    text (0.002, .07, 'Frame:' )
    text (0.003, .07, num2str(k))
pause(.05)

end



%% Noisy case
clear all; close all; clc; %Start Fresh

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

%% Initial Dimension Check
[A1_2,B1_2,C1_2,D1_2] = size(vidFrames1_2);
[A2_2,B2_2,C2_2,D2_2] = size(vidFrames2_2);
[A3_2,B3_2,C3_2,D3_2] = size(vidFrames3_2);
%% Resizing videos to 250 by 250 and 314 Frames
vidFrames1_2 = vidFrames1_2(146:395,251:500,:,:);
vidFrames2_2 = vidFrames2_2(76:325,201:450,:,28:341);
vidFrames3_2 = vidFrames3_2(151:400,251:500,:,1:314);
[A1_2,B1_2,C1_2,D1_2] = size(vidFrames1_2);
[A2_2,B2_2,C2_2,D2_2] = size(vidFrames2_2);
[A3_2,B3_2,C3_2,D3_2] = size(vidFrames3_2);
X1_2 = [];
Y1_2 = [];
X2_2 = [];
Y2_2 = [];
X3_2 = [];
Y3_2 = [];
%% Camera 1
for i = 1:D1_2
    img = rgb2gray(vidFrames1_2(:,:,:,i)); %Make every frame Black and White
    [Max, Ind] = max(img(:)); %Reshapes img to vector and findd the max value of img and ...
        where it is indexed
    [y1_2, x1_2] = ind2sub(size(img),Ind);
    X1_2 = [X1_2, x1_2];
    Y1_2 = [Y1_2, y1_2];
end
%% Camera 2
for i = 1:D2_2
    img = rgb2gray(vidFrames2_2(:,:,:,i));
    [Max, Ind] = max(img(:));
    [y2_2, x2_2] = ind2sub(size(img),Ind);
    X2_2 = [X2_2, x2_2];
    Y2_2 = [Y2_2, y2_2];
end
%% Camera 3
for i = 1:D3_2
    img = rgb2gray(vidFrames3_2(:,:,:,i));
    [Max, Ind] = max(img(:));
    [y3_2, x3_2] = ind2sub(size(img),Ind);
    X3_2 = [X3_2, x3_2];
    Y3_2 = [Y3_2, y3_2];
end
%% Plots for Noisy Case
figure()
subplot(3,1,1)
plot(X1_2,'b')
```

```
hold on;
plot(Y1_2, 'r', 'Linewidth',[2])
hold off;
xlabel('Frame')
xlim([0,230])
ylabel('Position')
ylim([0,350])
legend('Movement in x-axis','Movement in y-axis')
title('Cam 1')

subplot(3,1,2)
plot(X2_2,'b')
hold on;
plot(Y2_2, 'r', 'Linewidth',[2])
hold off;
xlabel('Frame')
xlim([0,230])
ylabel('Position')
ylim([0,350])
legend('Movement in x-axis','Movement in y-axis')
title('Cam 2')

subplot(3,1,3)
plot(X3_2,'b', 'Linewidth',[2])
hold on;
plot(Y3_2, 'r')
hold off;
xlabel('Frame')
xlim([0,230])
ylabel('Position')
ylim([0,350])
legend('Movement in x-axis','Movement in y-axis')
title('Cam 3')

%% Standardize
X1_2 = zscore(X1_2);
X2_2 = zscore(X2_2);
X3_2 = zscore(X3_2);
Y1_2 = zscore(Y1_2);
Y2_2 = zscore(Y2_2);
Y3_2 = zscore(Y3_2);
%%
%% SVD
X2 = [X1_2;Y1_2;X2_2;Y2_2;X3_2;Y3_2];
x = linspace(1,6,6);
t = linspace(1,314,314);
[m,n]=size(X2); % compute data size
mn = mean(X2,2); % compute mean for each row
X2 = X2-repmat(mn,1,n); % subtract mean
[u,s,v]=svd(X2'/sqrt(n-1)); % perform the SVD
sig = diag(s).^2; % produce diagonal variances
Yu = u*X2'; % produce the principal components projection
Yv = X2'*v;


%%
figure()
sig=diag(s);

subplot(3,1,1), plot(sig,'ko','Linewidth',[1.5])
axis([0 6 0 5000])
set(gca,'Fontsize',[13],'Xtick',[0  10 15 20 25])
title('Plot of Sigma')

subplot(3,1,2) % spatial modes
plot(x,u(:,1),'k','Linewidth',[2])
set(gca,'Fontsize',[13])
legend('mode 1','Location','NorthWest')
```

```
title('Spatial Modes')

subplot(3,1,3) % time behavior
plot(t,v(:,1),'k','Linewidth',[2])
set(gca,'Fontsize',[13])
title('Time Behavior')
%% Movie
figure()
for k = 1:D1_2
    ColorScale = (D1_2 - k)/D1_2;
    CSM = [ColorScale, ColorScale, ColorScale];
    Frame = k;
    plot(0, v(k,1),'.', 'MarkerSize', [10], 'color', [0 0 0])
    axis([-0.05 0.05 -0.1 0])
    set(gca,'Fontsize',[13],'Ytick',[],'Xtick',[])
    xlabel('Axis 1')
    ylabel('Axis 2')
    title('Movement in Relevant Axes')
    text (0.017, -0.025, 'Frame:' )
    text (0.025, -0.025, num2str(k))
pause(.05)

end



%% Horizontal Displacement
clear all; close all; clc; %Start Fresh

load('cam1_3.mat');
load('cam2_3.mat');
load('cam3_3.mat');

%% Initial Dimension Check
[A1_3,B1_3,C1_3,D1_3] = size(vidFrames1_3);
[A2_3,B2_3,C2_3,D2_3] = size(vidFrames2_3);
[A3_3,B3_3,C3_3,D3_3] = size(vidFrames3_3);
%% Resizing videos to 200 by 200 and 237 Frames
vidFrames1_3 = vidFrames1_3(201:400,241:440,:,3:239);
vidFrames2_3 = vidFrames2_3(161:360,226:425,:,39:275); % This video starts approximately ...
    18 frames earlier than Camera 1 and ends later as well
vidFrames3_3 = vidFrames3_3(151:350,276:475,:,:); % This video starts approximately 7
[A1_3,B1_3,C1_3,D1_3] = size(vidFrames1_3);
[A2_3,B2_3,C2_3,D2_3] = size(vidFrames2_3);
[A3_3,B3_3,C3_3,D3_3] = size(vidFrames3_3);
X1_3 = [];
Y1_3 = [];
X2_3 = [];
Y2_3 = [];
X3_3 = [];
Y3_3 = [];
%% Camera 1
for i = 1:D1_3
    img = rgb2gray(vidFrames1_3(:,:,:,i)); %Make every frame Black and White
    [Max, Ind] = max(img(:)); %Reshapes img to vector and findd the max value of img and ...
        where it is indexed
    [y1_3, x1_3] = ind2sub(size(img),Ind);
    X1_3 = [X1_3, x1_3];
    Y1_3 = [Y1_3, y1_3];
end
%% Cam 2 Pink to White
for j = 1:D2_3
rgbImage = vidFrames2_3(:,:,:,j);
%subplot(2, 1, 1);
%imshow(rgbImage);
%Extract the individual red, green, and blue color channels.
redChannel = rgbImage(:, :, 1);
greenChannel = rgbImage(:, :, 2);
```

```
blueChannel = rgbImage(:, :, 3);
%Get the yellow mask
pinkMask = 240 < redChannel & 170 < greenChannel & 200 < blueChannel;
% Make Pink mask White
redChannel(pinkMask) = 255;
greenChannel(pinkMask) = 255;
blueChannel(pinkMask) = 255;
% Recombine separate color channels into a single, true color RGB image.
rgbImage2 = cat(3, redChannel, greenChannel, blueChannel);
%subplot(2, 1, 2);
%imshow(rgbImage2);
vidFrames2_3(:,:,:,j) =  rgbImage2;
end

%% Camera 2
for i = 1:D2_3
    img = rgb2gray(vidFrames2_3(:,:,:,i));
    [Max, Ind] = max(img(:));
    [y2_3, x2_3] = ind2sub(size(img),Ind);
    X2_3 = [X2_3, x2_3];
    Y2_3 = [Y2_3, y2_3];
end

%% Camera 3
for i = 1:D3_3
    img = rgb2gray(vidFrames3_3(:,:,:,i));
    [Max, Ind] = max(img(:));
    [y3_3, x3_3] = ind2sub(size(img),Ind);
    X3_3 = [X3_3, x3_3];
    Y3_3 = [Y3_3, y3_3];
end
%% Plots for Horizontal Displacement
figure()
subplot(3,1,1)
plot(X1_3,'b')
hold on;
plot(Y1_3, 'r', 'Linewidth',[2])
hold off;
xlabel('Frame')
xlim([0,250])
ylabel('Position')
ylim([0,350])
legend('Movement in x-axis','Movement in y-axis')
title('Cam 1')

subplot(3,1,2)
plot(X2_3,'b')
hold on;
plot(Y2_3, 'r', 'Linewidth',[2])
hold off;
xlabel('Frame')
xlim([0,250])
ylabel('Position')
ylim([0,350])
legend('Movement in x-axis','Movement in y-axis')
title('Cam 2')

subplot(3,1,3)
plot(X3_3,'b', 'Linewidth',[2])
hold on;
plot(Y3_3, 'r')
hold off;
xlabel('Frame')
xlim([0,250])
ylabel('Position')
ylim([0,350])
legend('Movement in x-axis','Movement in y-axis')
title('Cam 3')
```

```
%% Standardize
X1_3 = X1_3/mean(X1_3);
X2_3 = X2_3/mean(X2_3);
X3_3 = X3_3/mean(X3_3);
Y1_3 = Y1_3/mean(Y1_3);
Y2_3 = Y2_3/mean(Y2_3);
Y3_3 = Y3_3/mean(Y3_3);
%% SVD
X3 = [X1_3;Y1_3;X2_3;Y2_3;X3_3;Y3_3];
x = linspace(1,6,6);
t = linspace(1,237,237);
[m,n]=size(X3); % compute data size
mn = mean(X3,2); % compute mean for each row
X3 = X3-repmat(mn,1,n); % subtract mean
[u,s,v]=svd(X3'/sqrt(n-1)); % perform the SVD
sig = diag(s).^2; % produce diagonal variances
Yu = u*X3'; % produce the principal components projection
Yv = X3'*v;


%%
figure()

subplot(3,1,1), plot(sig,'ko','Linewidth',[1.5])
 axis([0 6 0 5000])
 set(gca,'Fontsize',[13],'Ytick', [0 2500 5000],'Xtick',[0 1 2 3 4 5 6]);
 title('Plot of Sigma')

subplot(3,1,2) % spatial modes
plot(x,v(:,1),'k:', x, v(:,2),'k', 'Linewidth',[2])
set(gca,'Fontsize',[13])
legend('mode 1','mode 2','Location','NorthWest')
title('Spatial Modes')

subplot(3,1,3) % time behavior
plot(t,u(:,1),'k:',t,u(:,2),'k','Linewidth',[2])
set(gca,'Fontsize',[13])
title('Time Behavior')

% subplot(4,1,4) % time behavior
% plot(t,u(:,3),'k','Linewidth',[2])
% set(gca,'Fontsize',[13])
% title('Time Behavior')

%% 3d Plot of the change in x and y with respect to time
figure()
plot3(t, v(:,1),v(:,2),'k','Linewidth',[2])
xlabel('Frame')
ylabel('x(t)')
zlabel('y(t)')
grid on
%% Movie
figure()
for k = 1:D1_3
    plot(v(k,1), v(k,2),'.', 'MarkerSize', [10])
    axis([-0.5 0.5 -0.5 0.5])
pause(.05)
end
%% Movie Vid 3
figure()
for k = 1:D1_3
    ColorScale = (D1_3 - k)/D1_3;
    CSM = [ColorScale, 1-ColorScale , (1+ColorScale)/2];
    Frame = k;
    plot3(k, v(k,1),v(k,2),'.', 'MarkerSize', [10], 'color', [0 0 0])
    axis([1 237 -.085 -.035 -.15 .15])
    %set(gca,'Fontsize',[13],'Ytick',[],'Xtick',[])
    xlabel('Axis 1')
```

```matlab
    ylabel('Axis 2')
    title('Movement in Relevant Axes')
    grid on
    hold on
    %text (0.1, .1, 'Frame:' )
    %text (0.112,.1, num2str(k))

pause(.05)

end




%% Horizontal Displacement and Rotation:
clear all; close all; clc; %Start Fresh

load('cam1_4.mat');
load('cam2_4.mat');
load('cam3_4.mat');

%% Initial Dimension Check
[A1_4,B1_4,C1_4,D1_4] = size(vidFrames1_4);
[A2_4,B2_4,C2_4,D2_4] = size(vidFrames2_4);
[A3_4,B3_4,C3_4,D3_4] = size(vidFrames3_4);
%% Resizing videos to 200 by 200 and 392 Frames
vidFrames1_4 = vidFrames1_4(226:425,276:475,:,:);
vidFrames2_4 = vidFrames2_4(101:300,226:425,:,14:405);
vidFrames3_4 = vidFrames3_4(151:350,276:475,:,3:394);
[A1_4,B1_4,C1_4,D1_4] = size(vidFrames1_4);
[A2_4,B2_4,C2_4,D2_4] = size(vidFrames2_4);
[A3_4,B3_4,C3_4,D3_4] = size(vidFrames3_4);
X1_4 = [];
Y1_4 = [];
X2_4 = [];
Y2_4 = [];
X3_4 = [];
Y3_4 = [];
%% Cam 1 Remove non pink
for j = 1:D1_4
rgbImage = vidFrames1_4(:,:,:,j);
%Extract the individual red, green, and blue color channels.
redChannel = rgbImage(:, :, 1);
greenChannel = rgbImage(:, :, 2);
blueChannel = rgbImage(:, :, 3);
%Pink Mask
pinkMask = 245< redChannel &125< greenChannel& greenChannel < 195 & 140 <blueChannel & ...
    blueChannel<210;
%NonpinkMask
nonpinkMask = 240 > redChannel | 130 > greenChannel| greenChannel > 210 | 155 > ...
    blueChannel | blueChannel > 210;

% Remove Unwanted colors
redChannel (nonpinkMask) = 0;
greenChannel (nonpinkMask) = 0;
blueChannel (nonpinkMask) = 0;

% Recombine separate color channels into a single, true color RGB image.
rgbImage2 = cat(3, redChannel, greenChannel, blueChannel);
vidFrames1_4(:,:,:,j) =  rgbImage2;
end


%% Camera 1
for i = 1:D1_4
    img = rgb2gray(vidFrames1_4(:,:,:,i)); %Make every frame Black and White
    [Max, Ind] = max(img(:)); %Reshapes img to vector and findd the max value of img and ...
```

```
            where it is indexed
    [y1_4, x1_4] = ind2sub(size(img),Ind);
    X1_4 = [X1_4, x1_4];
    Y1_4 = [Y1_4, y1_4];
end

%% Camera 2
for i = 1:D2_4
    img = rgb2gray(vidFrames2_4(:,:,:,i));
    [Max, Ind] = max(img(:));
    [y2_4, x2_4] = ind2sub(size(img),Ind);
    X2_4 = [X2_4, x2_4];
    Y2_4 = [Y2_4, y2_4];
end

%% Camera 3
for i = 1:D3_4
    img = rgb2gray(vidFrames3_4(:,:,:,i));
    [Max, Ind] = max(img(:));
    [y3_4, x3_4] = ind2sub(size(img),Ind);
    X3_4 = [X3_4, x3_4];
    Y3_4 = [Y3_4, y3_4];
end
%% Plots for Horizontal Displacement and Rotation
figure()
subplot(3,1,1)
plot(X1_4,'b')
hold on;
plot(Y1_4, 'r', 'Linewidth',[2])
hold off;
xlabel('Frame')
xlim([0,230])
ylabel('Position')
ylim([0,350])
legend('Movement in x-axis','Movement in y-axis')
title('Cam 1')

subplot(3,1,2)
plot(X2_4,'b')
hold on;
plot(Y2_4, 'r', 'Linewidth',[2])
hold off;
xlabel('Frame')
xlim([0,230])
ylabel('Position')
ylim([0,350])
legend('Movement in x-axis','Movement in y-axis')
title('Cam 2')

subplot(3,1,3)
plot(X3_4,'b', 'Linewidth',[2])
hold on;
plot(Y3_4, 'r')
hold off;
xlabel('Frame')
xlim([0,230])
ylabel('Position')
ylim([0,350])
legend('Movement in x-axis','Movement in y-axis')
title('Cam 3')


%% Standardize
m1X=mean(X1_4);
m2X=mean(X2_4);
m3X=mean(X3_4);
m1Y=mean(Y1_4);
m2Y=mean(Y2_4);
```

```
m3Y=mean(Y3_4);
X1_4 = X1_4/m1X;
X2_4 = X2_4/m2X;
X3_4 = X3_4/m3X;
Y1_4 = Y1_4/m1Y;
Y2_4 = Y2_4/m2Y;
Y3_4 = Y3_4/m3Y;
%% Standardize
X1_4 = zscore(X1_4);
X2_4 = zscore(X2_4);
X3_4 = zscore(X3_4);
Y1_4 = zscore(Y1_4);
Y2_4 = zscore(Y2_4);
Y3_4 = zscore(Y3_4);
%% SVD
X4 = [X1_4;Y1_4;X2_4;Y2_4;X3_4;Y3_4];
x = linspace(1,6,6);
t = linspace(1,392,392);
[m,n]=size(X4); % compute data size
mn = mean(X4,2); % compute mean for each row
X4 = X4-repmat(mn,1,n); % subtract mean
[u,s,v]=svd(X4'/sqrt(n-1)); % perform the SVD
sig = diag(s).^2; % produce diagonal variances
Yu = u*X4'; % produce the principal components projection
Yv = X4'*v;
%%
figure()
subplot(3,1,1)
plot(t, Yv(:,1),'k')
subplot(3,1,2)
plot(t, Yv(:,2),'k')
subplot(3,1,3)
plot(t, Yv(:,3),'k')
%%
figure()

subplot(5,1,1), plot(sig,'ko','Linewidth',[1.5])
 axis([0 6 0 5000])
% set(gca,'Fontsize',[13],'Xtick',[0  10 15 20 25]);
 title('Plot of Sigma')

%  subplot(5,2,2), semilogy(sig,'ko','Linewidth',[1.5])
%  %axis([0 6 0 100])
%  %set(gca,'Fontsize',[13],'Ytick',[0, 10^1,10^2]);
%  title('Semi-log plot of Sigma')


subplot(5,1,2) % spatial modes
plot(x,v(:,1),'k--', x,v(:,2),'k:', x, v(:,3), 'k', 'Linewidth',[2])
set(gca,'Fontsize',[13])
legend('mode 1','mode 2', 'mode 3', 'Location','NorthWest')
 title('Spatial Modes')


subplot(5,1,3) % time behavior
plot(t,u(:,1),'k--','Linewidth',[2])
title('Time Behavior Mode 1')
subplot(5,1,4) % time behavior
plot(t,u(:,2),'k:','Linewidth',[2])
title('Time Behavior Mode 2')
subplot(5,1,5) % time behavior
plot(t,u(:,3),'k','Linewidth',[2])
title('Time Behavior Mode 3')

%%
figure()
for k = 1:D1_4
    plot(u(k,1),u(k,3),'.', 'MarkerSize', [10], 'color', [0 0 0])
```

```
    %axis([-.1 0 -.15 .1 -.1 .15])
    set(gca,'Fontsize',[13],'Ytick',[])
    xlabel('Y - Axis')
    ylabel('Z - Axis')
    %zlabel('Axis 3')
    title('Movement in Relevant Axes')
    grid on
    hold on
    %text (0.06, .1,.1, 'Frame:' )
    %text (0.06,.1,.1, num2str(k))
%pause(.03)

end

%% Movie Vid 4
figure()
for k = 1:D1_4
    ColorScale = (D1_4 - k)/D1_4;
    CSM = [ColorScale, 1-ColorScale , (1+ColorScale)/2];
    Frame = k;
    plot3(u(k,1),u(k,2),u(k,3),'.', 'MarkerSize', [10], 'color', CSM)
    axis([-.1 0 -.15 .1 -.1 .15])
    %set(gca,'Fontsize',[13],'Ytick',[],'Xtick',[])
    xlabel('Axis 1')
    ylabel('Axis 2')
    zlabel('Axis 3')
    title('Movement in Relevant Axes')
    grid on
    hold on
    %text (0.06, .1,.1, 'Frame:' )
    %text (0.06,.1,.1, num2str(k))
pause(.03)

end
```