AMATH 482 Homework 3

Jerry Wen

February 21st, 2021

Abstract

In this assignment, we will be discovering the uses of Principal Component Analysis and how it can help us track the different displacements of a mass on a spring. We'll be finding out how different Principal Components correspond to those specific movements and how the energy of the singular values will help us determine the most important information in the data to recreate our spring movement.

1 Introduction and Overview

We are given a video where we watch a spring-mass system bounces up and down. Through MATLAB, we will be finding the location of this mass in time. We will have 4 sets of videos, where each set contains 3 different angles of the mass moving up and down. Each set corresponds to a different case:

1. An Ideal Case: In this first case, all the cameras are fairly steady and the mass just simply goes up and down. This is to make a base case that we can compare the other cases to.
2. A Noisy Case: In this case, the mass continues to simply displace in the z axis, however noise is introduced in the videos by having each camera shake.
3. A Horizontal Displacement Case: In this case, the mass not only displaces up and down, but also moves side to side.
4. A Horizontal Displacement and Rotation Case: In this final case, the mass displaces up and down, moves side to side, and also rotates.

Using Singular Value Decomposition (SVD)/ Principal Component Analysis (PCA), we will be tracking the mass in each video and finding more details about our video.

2 Theoretical Background

In this assignment, we will be focusing on using SVD, specifically PCA. The SVD is just a way of factoring matricies, and PCA is a specific technique of using SVD.

Let us have a vector a, where $a = \{a_1, a_2, ..., a_n\}$ We can calculate the mean and variance of all the elements of a using simple formulas.

Calculating squared variance usually has each element get subtracted by the mean, however if we assume that has already been done, our formula matches the formula for the 2-norm squared which is just the inner/dot product:

$$\sigma^2 = \frac{1}{n}\sum_{k=0}^{n} a_k^2 = \frac{1}{n}||a||_2^2 = \frac{1}{n}aa^T \tag{1}$$

For an unbiased estimator, we use the formula $\frac{1}{n-1}aa^T$. With two vectors a and b, we can measure how much the variables vary with respect to each other. This is given by the formula

$$\sigma_{ab}^2 = \frac{1}{n-1}ab^T \tag{2}$$

This is called the covariance. If the covariance is positive, there is a positive relationship between the two variables. If the covariance is negative, there is an inverse relationship between the variables. If the covariance is zero, then there is no relationship between the variables.

Now, if there is multiple variables we want to look at, we can stack them together in a matrix X:

$$X = [a; b; c; d] \tag{3}$$

And to get the covariance we can multiply this matrix by its inverse:

$$C_x = \frac{1}{n-1} X X^T \tag{4}$$

This results in a symmetric square matrix, where a diagonal represents the variance of {a,b,c,d} and the other entries in the matrix so the covariance between two variables. However, as I talk about later in Computational results, we won't be needed the covariance matrix to get the information out of these videos. All we need to do is some pre-processing on our matrix: subtracting out the mean of each vector, as I discussed earlier that subtracting the mean is necessary to calculate the variance, and trimming each video so they all have the same amount of frames.

Once we combine all these vectors into a matrix X, taking the SVD seperates a matrix into three parts:

$$U, S, V$$

where $X = U * S * V'$ S is the main thing we focus on, as it is just a diagonal matrix that contains our singular values in descending order.

How large these singular values are compared to the others tells us how important that Nth mode of data is to recreate our original data. We can recreate a Nth mode matrix by multiplying the first N columns of U with the first N singular values from S and the first N columns of V prime.

$$\texttt{rank N matrix} = U(:, N) * S(1:N, 1:N) * V(:, 1:N)' \tag{5}$$

Where rank N is is the Nth mode estimation of the true matrix X.

To calculate the energies of each singular value, we have to normalize it so they all add up to 1. We have to square each singular value and divide it by the sum of each squared singular value:

$$sig = diag(S) \tag{6}$$

$$\texttt{energy calculation of singular values} = sig^2 / sum(sig^2) \tag{7}$$

In our project, the first two singular values are the more important part we are focusing on as it show the two most dominant motions, which should be up and down.

3 Algorithm Implementation and Development

Here is the main idea on how to implement and develop the code.

1. Start with three sets of data given by each angle taken by a camera. The datasets are 4-D matrices.
2. From here, we will crop the matrices in the x and y plane to mainly focus on the displacing mass. This will make it easier to locate the red marking on the top of the mass.
3. We then attempt to locate the max of each frame in the video. The max being the red marking on top of the mass. We will record the x and y position of this max in every frame for each camera.
4. With x and y position of all the angles, we will combine each vector to form a matrix to preform SVD on.
5. After completing the SVD, we can compare the energies of our singular values to see how much of a certain movement the mass does.

Algorithm 1: Sigma Energy Algorithm

Import data from `camn_m.mat` where n represents the camera number and m represents what case we are looking into
Crop the video so it only focuses on the displacing mass
Create the 2-D movement matrix of the mass for cameras 1-3
for $j = 1 :$ `number of total frames` **do**
  Take the frame j
  Find the max of this frame
  Add the x and y coordinate of this max to a separate matrix
end for
We do this loop for each camera
Separate the x and y coordinate to create their own unique vectors from each camera
Subtract the average from all 6 of these vectors to prepare for PCA
Trim the length of these vectors so they all start and end on the same frame
Combine these vectors into a 6xN matrix and perform SVD
Get the singular values from sigma and plot the energies of each singular value

4 Computational Results

Our results come in the form of graphical representations of the values of the singular values that we got from each Case.

We know that if a camera is not shot in the right angle, the first principal component will tell us that we need to rotate it. Because we know that one of the cameras is positioned incorrectly, I looked into creating a "fixed" version of it where camera 3 is rotated. Such rotation was done by taking the x and y vector and multiplying it by the inverse Rotation matrix by 90 degrees, as we needed to rotate it clockwise.

However, if we compare the Sigma energies of the non-rotated matrix and the rotated matrix, they come out to be the same:

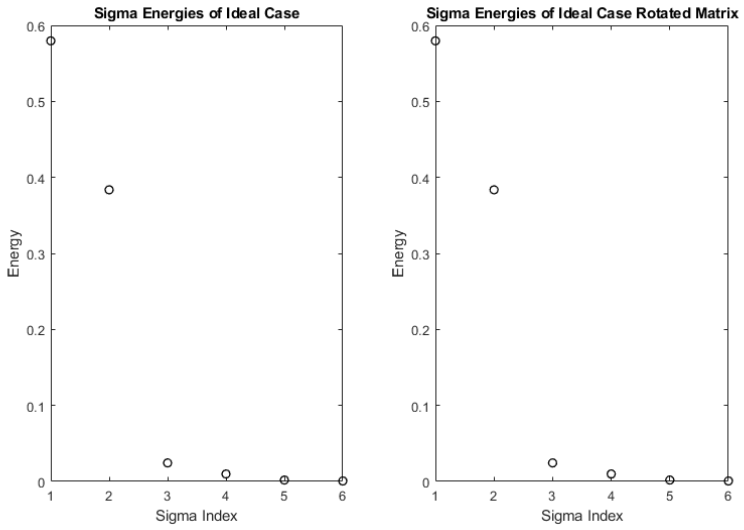
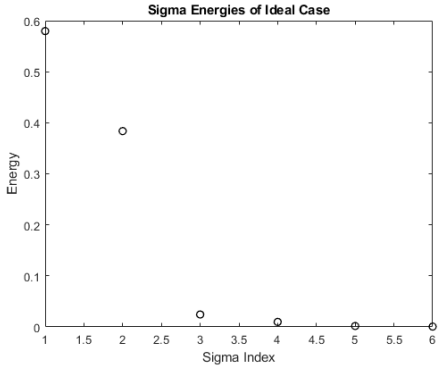


Figure 1: A comparison of the Sigma Energies of the Non-Rotated and Rotated Matrix

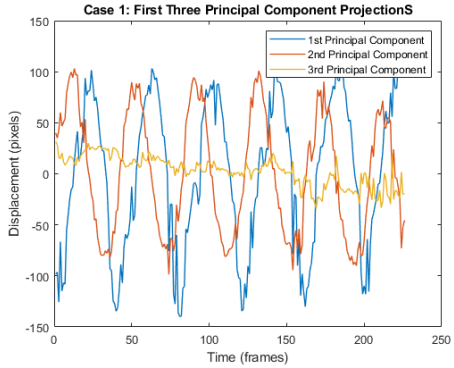
With this knowledge, I realized the just computing the SVD of the original matrix that has all the vectors combined will do the rotation for us as it MATLAB will be able to calculate the primary components and check if there are any negative values in the first primary component.

4.1 Case 1: The Ideal Case

Here are the Sigma Energies for the first case in higher detail:



(a) The Sigma Energies of Case 1



(b) The First Three Principal Components Projection

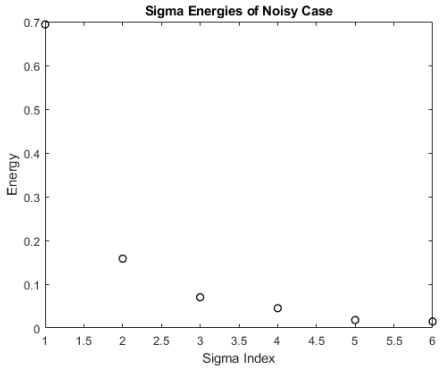
Figure 2: Information for Case 1

Here we see that the first two singular values are significantly higher than the following singular values, with a energy value of 0.5796 and 0.3837. We expected this as this is our Ideal Case and the videos are held still enough to not generate too much noise and the first two singular values represent upwards and downwards movement. If this was a perfect Ideal Case, the first singular value should be a bit more than 0.5 and the second would be a bit less than 0.5, but they would add up to equal 1, making the other singular values 0.

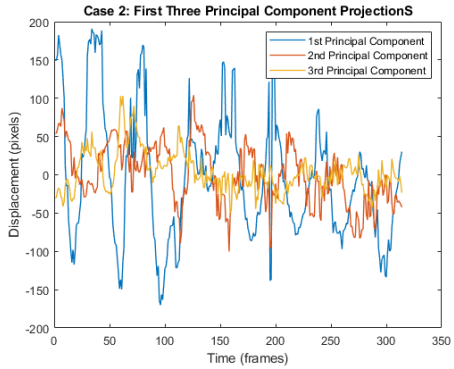
In the Principal Components Projection Graph(fig:2b) We see that the first and second principal component contributes to most of the movement/ looks the most like sin/cos graphs. The 3rd component and onwards which I have left out are mostly uninteresting.

4.2 Case 2: The Noisy Case

Here are the Sigma Energies calculated from the second case.



(a) The Sigma Energies of Case 2



(b) The First Three Principal Components Projection

Figure 3: Information for Case 2

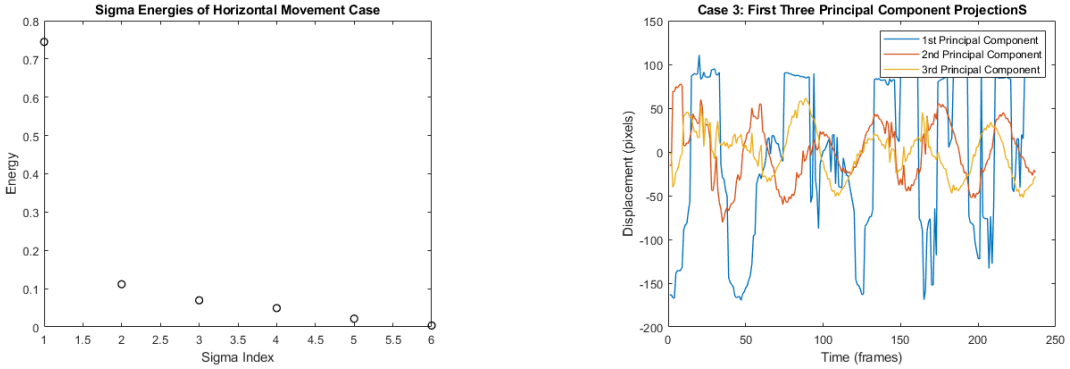
The resulting graph differs from the ideal case by a lot. We see that the total energy value of the first two singular value decreased: it went from a sum of 0.9633 from the Ideal Case(Fig:2) to a sum of 0.852 in this Noisy Case(Fig:3). The second singular value decreased a lot more while the first singular value increased. Because the cameras were shaking around so much, it was probably harder to track one of its two main

motions and probably made it look like it was moving side to side a bit as well as the other singular values also increased in energy.

In the principal component projection, we see that the second principal component doesn't have as much movement to it, however the third principal component is starting to jump around. This is the result of the noise; we are unable to see specific movements.

4.3 Case 3: Horizontal Displacement Case

Now, let's investigate how actual horizontal displacement along with vertical displacement changes the the values we get after using PCA on our data. Here are the Sigma Energies for the third case:



(a) The Sigma Energies of Case 3

(b) The First Three Principal Components Projection

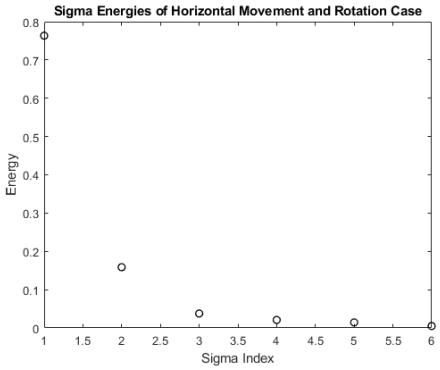
Figure 4: Information for Case 3

We can see, that this case actually differs equally from the Ideal Case(Fig:2) as the Noisy Case(Fig:3). The energy total of the first two singular values adds up to be 0.8561, which is comparable to Case 2's total of 0.852. It seems like we were able to detect this horizontal displacement, however the PCA determines that there about equal horizontal movement in this case as the Noisy Case(Fig:3).

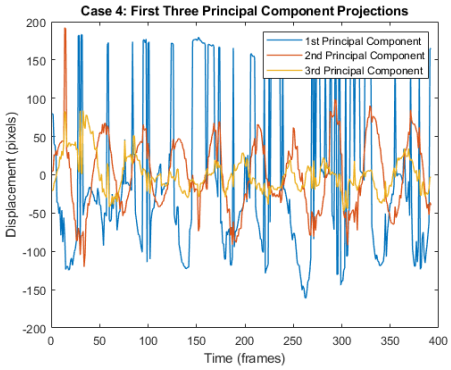
In the Principal Components Projection Graph, we can start to clearly see a sin wave being formed by the third singular value. This is expected as it is displacing in a different direction that is not down or up. Looking more into the next few Principal Components (Shown in Appendix C Fig:6), we can see that Principal Component 4 is also making a sin wave, but Principal Component 5 is not. This allows us to conclude that Component 3 and 4 are Horizontal Movement components and because we have incorporated the movement to the mass, we are able to see these sin waves present.

4.4 Case 4: Horizontal Displacement and Rotation Case

Now expanding upon Case 3, we are making the mass rotate as well. Here are the Sigma Energies for the fourth case:



(a) The Sigma Energies of Case 3



(b) The First Three Principal Components Projection

Figure 5: Information for Case 4

In this case, we can see it doesn't differ much from Case 3(Fig:4). However, the total energy of the first two singular values sums to 0.9222, which is actually right below the total energy of Case 1(Fig:2). This may be because the recording might have just been a bit better in stability, or the mass bounces up and down more due to the rotation it is doing.

Looking at the Principal Components Projection, we see very large jumps in the first and second Principal Component. However looking at Principal Components 3-5 (Shown in Appendix C Fig: 7), we are able to see that Component 5 is actually forming a somewhat clear sin wave as well, which makes me thing that it corresponds to the rotational aspect of the mass, maybe because it looks like it is getting close to the camera in the z axis.

5 Summary and Conclusions

From our results, we see that we were able to see that the Ideal Case's first singular value energy exceeds all the other cases as we expected. The Noisy Case came out to have to smallest energy value for the first singular value. We also saw that rotating the mass did not do much to change the information that we got from the video.

Through this project, we have seen the power of using SVD and PCA to turn multiple dimensions of data into information that we can digest.

Appendix A MATLAB Functions

- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix `A`, such that `A = U*S*V'`.
- `[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.
- `x = diag(A)` returns a column vector of the main diagonal elements of `A`

Appendix B MATLAB Code

```
%% Test 1: Ideal Case
clear; close all; clc;

load('cam1_1.mat')
% implay(vidFrames1_1)
load('cam2_1.mat')
% implay(vidFrames2_1)
load('cam3_1.mat')
% implay(vidFrames3_1)


cropVidFrames1_1 = vidFrames1_1(:,275:400,:,:);
% implay(cropVidFrames1_1)
cropVidFrames2_1 = vidFrames2_1(:,200:370,:,:);
% implay(cropVidFrames2_1)
cropVidFrames3_1 = vidFrames3_1(210:350,:,:,:);
% implay(cropVidFrames3_1)



numFrames_1 = size(cropVidFrames1_1,4);
cam1 = zeros(numFrames_1,2);
for i = 1:numFrames_1
    t1 = cropVidFrames1_1(:,:,1,i);
    [Max,Ind] = max(t1(:));
    [x1,y1] = ind2sub([size(t1,1), size(t1,2)], Ind);
    cam1(i,:) = [x1,y1];
end

numFrames_2 = size(cropVidFrames2_1,4);
cam2 = zeros(numFrames_2,2);
for i = 1:numFrames_2
    t2 = cropVidFrames2_1(:,:,1,i);
    [Max,Ind] = max(t2(:));
    [x1,y1] = ind2sub([size(t2,1), size(t2,2)], Ind);
    cam2(i,:) = [x1,y1];
end

numFrames_3 = size(cropVidFrames3_1,4);
cam3 = zeros(numFrames_3,2);
for i = 1:numFrames_3
    t3 = cropVidFrames3_1(:,:,1,i);
    [Max,Ind] = max(t3(:));
    [x1,y1] = ind2sub([size(t3,1), size(t3,2)], Ind);
    cam3(i,:) = [x1,y1];
end

% Set mean of the results equal to 0
cam1x = cam1(:,1) - mean(cam1(:,1));
cam1y = cam1(:,2) - mean(cam1(:,2));
cam2x = cam2(:,1) - mean(cam2(:,1));
cam2y = cam2(:,2) - mean(cam2(:,2));
cam3x = cam3(:,1) - mean(cam3(:,1));
cam3y = cam3(:,2) - mean(cam3(:,2));

% figure(1)
% plot(cam1x)
% hold on
```

```
% trim the vectors so they are the same length
trimLen = min([length(cam1x), length(cam2x), length(cam3x)]);
trimCam1x = cam1x(1:trimLen);
trimCam1y = cam1y(1:trimLen);
trimCam2x = cam2x(1:trimLen);
trimCam2y = cam2y(1:trimLen);
trimCam3x = cam3x(1:trimLen);
trimCam3y = cam3y(1:trimLen);

% Perform the Principal Component Analysis
firstMat = [trimCam1x trimCam1y trimCam2x trimCam2y trimCam3x trimCam3y]';
% SVD of X will give you one singular value if it is perfect data

[U,S,V] = svd(firstMat, 'econ');
sig = diag(S);

figure(1)
% subplot(1,2,1)
plot(sig.^2/sum(sig.^2),'ko','Linewidth',1)
xlabel('Sigma Index'); ylabel('Energy')
title('Sigma Energies of Ideal Case')

% prin_comp = U' * firstMat;
% plot(1:trimLen, prin_comp(1:3,:),'Linewidth',1)
% xlabel('Time (frames)'); ylabel('Displacement (pixels)')
% legend('1st Principal Component', '2nd Principal Component', ...
%        '3rd Principal Component', 'location', 'northeast')
% title('Case 1: First Three Principal Component ProjectionS')
%% Test 2: Noisy Case
clear; clc;
load('cam1_2.mat')
load('cam2_2.mat')
load('cam3_2.mat')

% implay(vidFrames1_2)
cropVidFrames1_2 = vidFrames1_2(:,300:450,:,:);
% implay(cropVidFrames1_2)
cropVidFrames2_2 = vidFrames2_2(:,180:450,:,:);
% implay(cropVidFrames2_2)
cropVidFrames3_2 = vidFrames3_2(175:350,:,:,:);
% implay(cropVidFrames3_2)

numFrames_1 = size(cropVidFrames1_2,4);
cam1 = zeros(numFrames_1,2);
for i = 1:numFrames_1
    t1 = cropVidFrames1_2(:,:,1,i);
    [Max,Ind] = max(t1(:));
    [x1,y1] = ind2sub([size(t1,1), size(t1,2)], Ind);
    cam1(i,:) = [x1,y1];
end

numFrames_2 = size(cropVidFrames2_2,4);
cam2 = zeros(numFrames_2,2);
for i = 1:numFrames_2
    t2 = cropVidFrames2_2(:,:,1,i);
    [Max,Ind] = max(t2(:));
    [x1,y1] = ind2sub([size(t2,1), size(t2,2)], Ind);
    cam2(i,:) = [x1,y1];
end
```

```
numFrames_3 = size(cropVidFrames3_2,4);
cam3 = zeros(numFrames_3,2);
for i = 1:numFrames_3
    t3 = cropVidFrames3_2(:,:,1,i);
    [Max,Ind] = max(t3(:));
    [x1,y1] = ind2sub([size(t3,1), size(t3,2)], Ind);
    cam3(i,:) = [x1,y1];
end

cam1x = cam1(:,1) - mean(cam1(:,1));
cam1y = cam1(:,2) - mean(cam1(:,2));
cam2x = cam2(:,1) - mean(cam2(:,1));
cam2y = cam2(:,2) - mean(cam2(:,2));
cam3x = cam3(:,1) - mean(cam3(:,1));
cam3y = cam3(:,2) - mean(cam3(:,2));

trimLen = min([length(cam1x), length(cam2x), length(cam3x)]);
trimCam1x = cam1x(1:trimLen);
trimCam1y = cam1y(1:trimLen);
trimCam2x = cam2x(1:trimLen);
trimCam2y = cam2y(1:trimLen);
trimCam3x = cam3x(1:trimLen);
trimCam3y = cam3y(1:trimLen);

secondMat = [trimCam1x trimCam1y trimCam2x trimCam2y trimCam3x trimCam3y]';
[U,S,V] = svd(secondMat, 'econ');
sig = diag(S);
figure(2)
plot(sig.^2/sum(sig.^2),'ko','Linewidth',1)
xlabel('Sigma Index'); ylabel('Energy')
title('Sigma Energies of Noisy Case')

% prin_comp = U' * secondMat;
% plot(1:trimLen, prin_comp(1:3,:),'Linewidth',1)
% xlabel('Time (frames)'); ylabel('Displacement (pixels)')
% legend('1st Principal Component', '2nd Principal Component', ...
%        '3rd Principal Component', 'location', 'northeast')
% title('Case 2: First Three Principal Component ProjectionS')

%% Test 3: Horizontal Displacement
clear; clc;
load('cam1_3.mat')
load('cam2_3.mat')
load('cam3_3.mat')


cropVidFrames1_3 = vidFrames1_3(:,250:400,:,:);
% implay(cropVidFrames1_3)
cropVidFrames2_3 = vidFrames2_3(:,180:450,:,:);
% implay(cropVidFrames2_3)
cropVidFrames3_3 = vidFrames3_3(150:350,:,:,:);
% implay(cropVidFrames3_3)

numFrames_1 = size(cropVidFrames1_3,4);
cam1 = zeros(numFrames_1,2);
for i = 1:numFrames_1
    t1 = cropVidFrames1_3(:,:,1,i);
    [Max,Ind] = max(t1(:));
    [x1,y1] = ind2sub([size(t1,1), size(t1,2)], Ind);
    cam1(i,:) = [x1,y1];
end
```

```
numFrames_2 = size(cropVidFrames2_3,4);
cam2 = zeros(numFrames_2,2);
for i = 1:numFrames_2
    t2 = cropVidFrames2_3(:,:,1,i);
    [Max,Ind] = max(t2(:));
    [x1,y1] = ind2sub([size(t2,1), size(t2,2)], Ind);
    cam2(i,:) = [x1,y1];
end

numFrames_3 = size(cropVidFrames3_3,4);
cam3 = zeros(numFrames_3,2);
for i = 1:numFrames_3
    t3 = cropVidFrames3_3(:,:,1,i);
    [Max,Ind] = max(t3(:));
    [x1,y1] = ind2sub([size(t3,1), size(t3,2)], Ind);
    cam3(i,:) = [x1,y1];
end

cam1x = cam1(:,1) - mean(cam1(:,1));
cam1y = cam1(:,2) - mean(cam1(:,2));
cam2x = cam2(:,1) - mean(cam2(:,1));
cam2y = cam2(:,2) - mean(cam2(:,2));
cam3x = cam3(:,1) - mean(cam3(:,1));
cam3y = cam3(:,2) - mean(cam3(:,2));

trimLen = min([length(cam1x), length(cam2x), length(cam3x)]);
trimCam1x = cam1x(1:trimLen);
trimCam1y = cam1y(1:trimLen);
trimCam2x = cam2x(1:trimLen);
trimCam2y = cam2y(1:trimLen);
trimCam3x = cam3x(1:trimLen);
trimCam3y = cam3y(1:trimLen);

thirdMat = [trimCam1x trimCam1y trimCam2x trimCam2y trimCam3x trimCam3y]';
[U,S,V] = svd(thirdMat, 'econ');
sig = diag(S);
figure(3)
plot(sig.^2/sum(sig.^2),'ko','Linewidth',1)
title('Sigma Energies of Horizontal Movement Case')
xlabel('Sigma Index'); ylabel('Energy')

% prin_comp = U' * thirdMat;
% plot(1:trimLen, prin_comp(3:5,:),'Linewidth',1)
% xlabel('Time (frames)'); ylabel('Displacement (pixels)')
% % legend('1st Principal Component', '2nd Principal Component', ...
% %        '3rd Principal Component', '4th Principal Component', ...
% %        '5th Principal Component', 'location', 'southeast')
%  legend('3rd Principal Component', '4th Principal Component', ...
%         '5th Principal Component', 'location', 'southeast')
% title('Case 3: Three to Five Principal Component ProjectionS')

%% Part 4: Horizontal Displacement and Rotation
%clear; close all; clc;
clear; clc;
load('cam1_4.mat')
load('cam2_4.mat')
load('cam3_4.mat')

cropVidFrames1_4 = vidFrames1_4(:,300:500,:,:);
% implay(cropVidFrames1_4)
```

```
cropVidFrames2_4 = vidFrames2_4(:,180:450,:,:);
% implay(cropVidFrames2_4)
cropVidFrames3_4 = vidFrames3_4(150:350,:,:,:);
% implay(cropVidFrames3_4)

numFrames_1 = size(cropVidFrames1_4,4);
cam1 = zeros(numFrames_1,2);
for i = 1:numFrames_1
    t1 = cropVidFrames1_4(:,:,1,i);
    [Max,Ind] = max(t1(:));
    [x1,y1] = ind2sub([size(t1,1), size(t1,2)], Ind);
    cam1(i,:) = [x1,y1];
end

numFrames_2 = size(cropVidFrames2_4,4);
cam2 = zeros(numFrames_2,2);
for i = 1:numFrames_2
    t2 = cropVidFrames2_4(:,:,1,i);
    [Max,Ind] = max(t2(:));
    [x1,y1] = ind2sub([size(t2,1), size(t2,2)], Ind);
    cam2(i,:) = [x1,y1];
end

numFrames_3 = size(cropVidFrames3_4,4);
cam3 = zeros(numFrames_3,2);
for i = 1:numFrames_3
    t3 = cropVidFrames3_4(:,:,1,i);
    [Max,Ind] = max(t3(:));
    [x1,y1] = ind2sub([size(t3,1), size(t3,2)], Ind);
    cam3(i,:) = [x1,y1];
end

cam1x = cam1(:,1) - mean(cam1(:,1));
cam1y = cam1(:,2) - mean(cam1(:,2));
cam2x = cam2(:,1) - mean(cam2(:,1));
cam2y = cam2(:,2) - mean(cam2(:,2));
cam3x = cam3(:,1) - mean(cam3(:,1));
cam3y = cam3(:,2) - mean(cam3(:,2));

trimLen = min([length(cam1x), length(cam2x), length(cam3x)]);
trimCam1x = cam1x(1:trimLen);
trimCam1y = cam1y(1:trimLen);
trimCam2x = cam2x(1:trimLen);
trimCam2y = cam2y(1:trimLen);
trimCam3x = cam3x(1:trimLen);
trimCam3y = cam3y(1:trimLen);


fourthMat = [trimCam1x trimCam1y trimCam2x trimCam2y trimCam3x trimCam3y]';
[U,S,V] = svd(fourthMat, 'econ');
sig = diag(S);
figure(4)
% plot(sig.^2/sum(sig.^2),'ko','Linewidth',1)
% title('Sigma Energies of Horizontal Movement and Rotation Case')
% xlabel('Sigma Index'); ylabel('Energy')
```

```
prin_comp = U' * fourthMat;
plot(1:trimLen, prin_comp(3:5,:),'Linewidth',1)
xlabel('Time (frames)'); ylabel('Displacement (pixels)')
legend('3rd Principal Component', '4th Principal Component', ...
       '5th Principal Component', 'location', 'northeast')
title('Case 4: Three to Five Principal Component Projections')
```

Appendix C Additional Graphs

Additional Graphs that are not needed but for additional information for those curious.

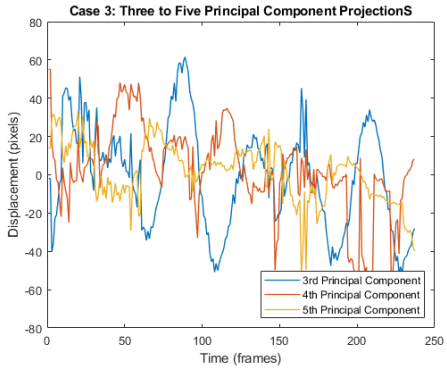


Figure 6: Principal Components 3 to 5 of Case 3

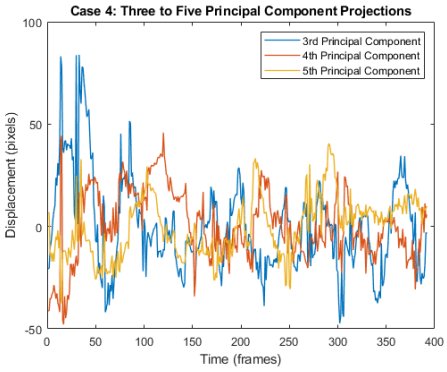


Figure 7: Principal Components 3 to 5 of Case 4