AMATH 482 Homework 3
Principle Component Analysis

Weien Guo

February 24, 2021

Abstract

Principle component is a very useful tool in analyzing high dimensional data. In this assignment, we are given 12 video data files, which are the recordings of a spring-mass system filmed from 3 different angles. We will explore the practical usefulness and the effects of noise on the PCA algorithms.

1 Introduction and Overview

An experiment with a spring-mass system is performed and recorded. A mass is hanging from a spring, and in this experiment, we use a white paint can with a white flash light attached at its top. The mass moves up and down due to the force in the spring and gravity. We will analyze this one-dimensional movement using singular value decomposition and principle component analysis. There are 3 cameras filming the experiment from different angles, and the films do not start and end at the same time. Worth noticing is that from the first two angles, the gravitational force is pointing downwards, and from the third angle, the force points to the right, meaning the camera turned 90 degrees to the left. The experiment is performed for 4 times, with different conditions:

1. **Test 1 Ideal Case**

   The mass moves in the z direction and the video has little to no oscillation.

2. **Test 2 Noisy Case**

   The movement of the mass remains the same as in test 1, but the cameras shake while recording.

3. **Test 3 Horizontal Displacement**

   The mass is released off-center, creating a motion in the x-y plane in addition to the z direction.

4. **Test 4 Horizontal Displacement and Rotation**

   The mass moves in the same way as in test 3, but this time the can rotates as well.

From each frame of the 12 recordings, we will need to extract the position of the paint can first before we can use the PCA algorithm. To get the clean position data, we will also need to filter out the background, and align the movement for further analysis.

2 Theoretical Background

The vertical displacement of the paint can in this spring-mass system can be calculated by the equation

$$z(t) = Acos(\omega t + \varphi)$$

where A and φ are determined by the initial displacement and the velocity of the mass. This equation is the solution of a second order differential equation of displacement from Newton's second law and Hooke's law. The parameter ω can be found from the ODE.

The position data is gathered from the recording, and then singular value decomposition is performed on the data.

2.1 Singular Value Decomposition

SVD is a very powerful and useful tool in computational mathematics. A matrix, representing a transformation from $\mathbb{R}^n$ to $\mathbb{R}^m$, can be decomposed to

$$A = U\Sigma V^*$$

with $U \in \mathbb{R}^{n\times m}$ and $V \in \mathbb{R}^{n\times n}$ are unitary, $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal.

The values on the diagonal of Σ, σ_i, are called singular values of the matrix A. Column in U are left singular values of A. Columns in V are right singular values of A. They also satisfy that $Av_i = \sigma_i u_i$ and $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n$. σ_i are always non-negative and real numbers. The number of non-zero singular values are called the rank of A. MATLAB has built-in function `svd()` that calculates U, Σ, V matrices for us.

2.2 Principle Component Analysis

Since U is unitary, it is an orthogonal matrix and vectors in the direction of each column are perpendicular to each other. The length u_i decreases as i increases. That is because the lengths are proportional to the spread of data in that direction. These perpendicular vectors are also called the principle components of the data. The orthogonality means that each variable contains completely new information and the greater the variances is, the variable contains more important information about the data. When performing PCA, the row mean of the input matrix needs to be zero.

The variance and co-variance matrix of the data matrix is equal to

$$C_X = \frac{1}{n-1} XX^T = AA^T = U\Sigma^2 U^T.$$

Calculating energy for helps us to determine what percentage of information about the entire data is included in each components, the formula is given as

$$energy_N = \frac{\|X_N\|^2}{X^2}$$

.

We can use the diagonal of Σ as X in this formula.

2.3 Image Processing

In this assignment, we work with images that are of the size 480×640. That means the image has 480 pixels in the vertical direction and 640 pixels in the horizontal direction. Each pixel is has a red, green, and blue component. The component ranges from 0 to 255, where 0 is the darkest (black), and 255 is the brightest (white). So in MATLAB, an image is represented as a $480 \times 640 \times 3$ array. In the analysis, we will turn each colored frame into gray scale, which means the image is now a $480 \times 640 \times 1$ matrix. When the value is 255, the pixel is white, and when the value is 0, the pixel is black. Therefore, the flash on top of the paint can can help us greatly in identifying the position of the mass because it creates a white spot that will give a very large number. Sometimes, the white wall or the reflection of light from some metal may cause the lost of accuracy, so we can use a Shannon filter, which sets unwanted pixels to black. In this way, we minimize the chance for getting a wrong mass position.

3 Algorithm Implementation and Development

1. Load the video data files (e.g. `cam1_1.mat`). The `implay()` command helps to view the video.

2. Because we will turn each frame to gray scale and find the maximum value, that is the location of the flash light sitting on top of the paint can, we need to use a Shannon filter to make unwanted pixels black, as in Figure 1. So we create a filter that leaves a portion of the entire frame visible, and the rest black. The small window is large enough that the paint can (the flash light) is not covered in all frames.

Figure 1: The effect of the Shannon filter for camera 1 in Test 1

3. In a for loop, we turn each frame to gray scale, apply the filter find the max point and store it in a vector. For the first test, because there are not a lot of shaking/noise, I observed that the white spot from the flash light is not just one pixel, and the `max()` function in MATLAB only returns the first maximum and its location. It would be better if we can get all the pixels of the brightest points and take the mean of them to get a more accurate location on the frame. So I applied this idea in my algorithm using `find()` function. In the second test, the noisy is very large, blurring the brightest spot in the frame. If we take the average, it would not be as accurate as in test 1 anymore. So for test 2, I only used the maximum. Similar in test 3, but this time, the fast movement blurs the white spot. I also used the maximum only. In test 4, the movement slows down and film is steady. So we are again able to average the position of the brightest points.

4. Because the videos start and end at different times, we would want to trim them and align the movement of the can. This is done by making the video from each of the 3 angles start at the first turning point of the mass, that is the maximum in the first 50 frames. After the videos from the angles go through the above steps, we trim the vectors using the shortest length among the, and put them together in one matrix `X`. We also need to make sure that the rows of this matrix have zero mean.

5. Using the `svd()` function in MATLAB to perform the singular value decomposition for us. `U, S, V` are the returned matrices from the `svd()` command, and we use them for visualizations.

4 Computational Results

Test 1: Ideal case

Figure 2 shows the position of the paint can in the x-y coordinate system after using the algorithm. We can clearly see from the first two plots that the motion of the can is almost purely in one-direction, that is in the vertical direction. That is because of the gravitational force is the largest factor in this test.

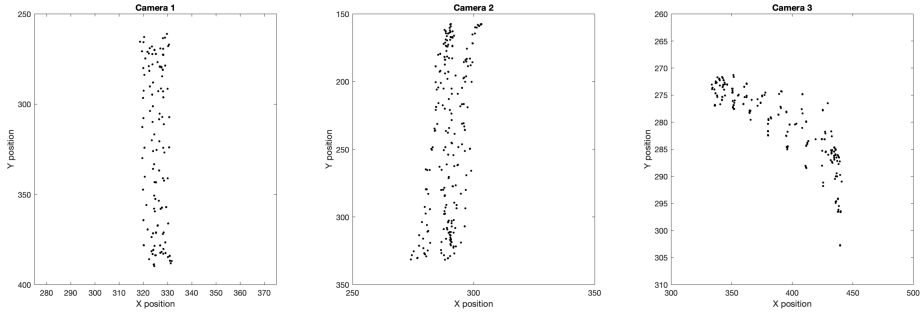

Figure 2: Position of the can from 3 angles in Test 1

Figure 3 is the result from running the SVD algorithm. In this test, there is one dominant component, that captures about 78% of the energy. In other words, this component contains about 78% of the variance

in the data. The second component contains about 10%. The other 4 components contain relatively much less energy. The middle plot shows the first two columns of the matrix `V`. The blue line is the paint can movement as projected onto the first principle component. The motion is clearly sinusoidal, and the red line, which is the second column of `V` has lots of noise. That means the first component is able to explain the movement of the can very well. The plot on the right is the variance of the movement, calculated using $V \times S$. Other than the first component, the others does not show a clear pattern. So, we can again conclude that the first component would give us very good explanation of the data. This conclusion agrees with our expectation, since the force involved in this experiment is mainly gravity.

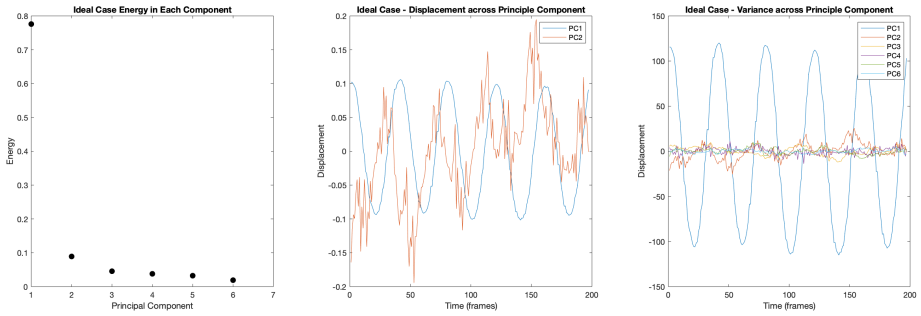

Figure 3: Energy, displacement, and variance results for Test 1

Test 2: Noisy case

Figure 4 shows the position of the paint can in the x-y coordinate system after using the algorithm. Because of the intense shaking in the recording, there is lots of oscillation in the data. As in the middle plot, other than the vertical variance, the variance in the horizontal direction is also considerable, although there was not any force in that direction.

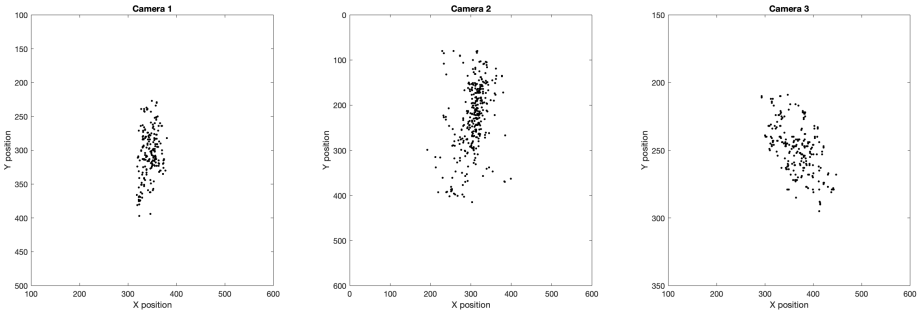

Figure 4: Position of the can from 3 angles in Test 2

Figure 5 is the result from running the SVD algorithm. In this test, the dominant component contains about 41% of the energy. This is much less as we compare it with that from test 1. The second component contains about 20%, meaning a rank-2 approximation would only explain about 61% of the variance in the data. From the middle and right plots, first two, or even three components can all reveal some information about the movement of the can. Therefore, we have to say that the noisiness in the recording has a huge impact on our principle component analysis. We would need much more components to explain the movement of the can in these recordings.

Test 3: Horizontal Displacement

Figure 6 shows the position of the paint can in the x-y coordinate system after using the algorithm. The horizontal displacement is shown clearly in the middle and right plots. When the horizontal force combines with the vertical force from gravity and the spring, it results in a circular motion going up and down. We would expect to need at least two components to explain the movement.

Figure 7 is the result from running the SVD algorithm. In this test, the first component contains about

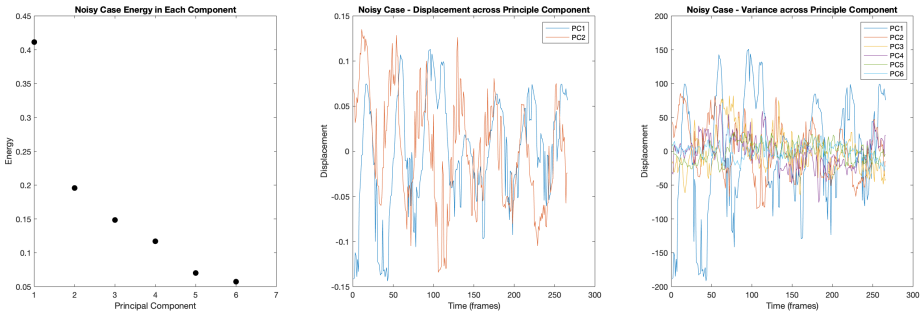

Figure 5: Energy, displacement, and variance results for Test 2

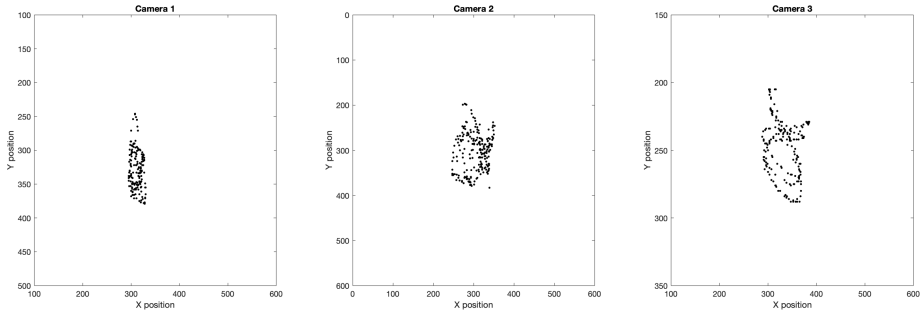

Figure 6: Position of the can from 3 angles in Test 3

38% of the energy, the second contains about 25%, and the third contains about 17%. Most of the energy is distributed in the first few components, which is different from the previous two tests. The displacement from the first and second components both show a sinusoidal movement, although not very smooth, meaning that probably more component is needed to explain the movement. The yellow line in the right plot is the variance from the third component. It also shows some ability in explaining the variance.

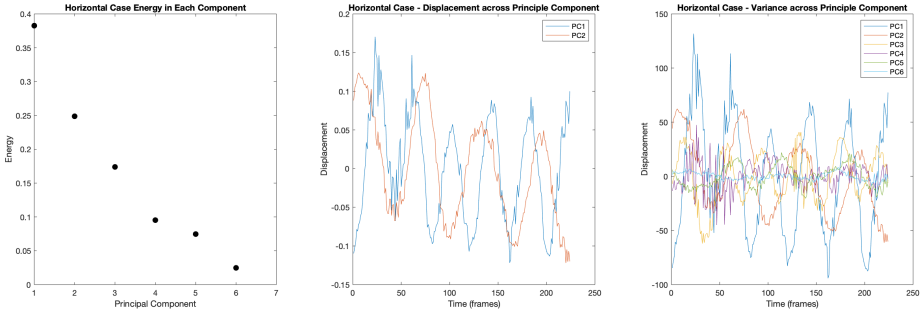

Figure 7: Energy, displacement, and variance results for Test 3

Test 4: Horizontal Displacement and Rotation

Figure 8 shows the position of the paint can in the x-y coordinate system after using the algorithm. In this test, the can was released off-center, but the horizontal movement disappeared in the later half of the recording and the movement is vertical with rotation.

Figure 9 is the result from running the SVD algorithm. In this test, the first component contains about 51% of the energy, the second contains about 18%, and the third contains about 13%. Most of the energy is distributed in the first few components. The first component could explain about half of the variance in the data. That is probably because the vertical movement in the second half averaged out some of the horizontal

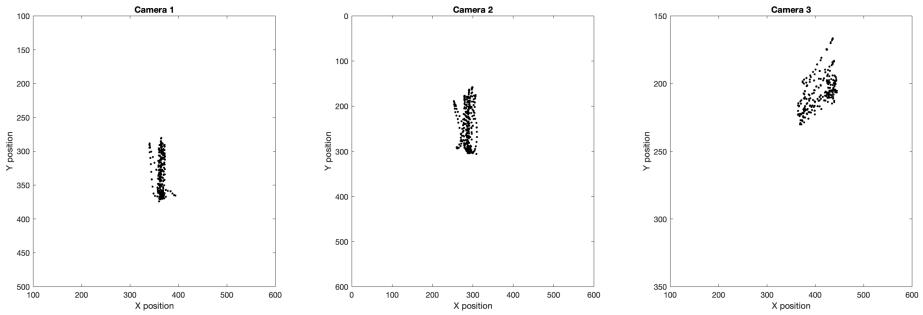

Figure 8: Position of the can from 3 angles in Test 4

movement. From the middle plot, we can see that the first two components both show a clear sinusoidal shape. The third component also has some pattern in the right plot,so all three of them are able to explain some information about the can movement.

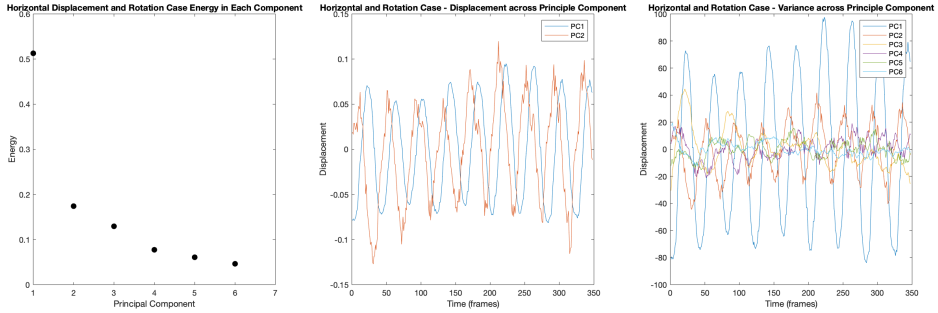

Figure 9: Energy, displacement, and variance results for Test 4

5 Summary and Conclusions

In this assignment, we used principle component analysis to interpret the movement of the mass in a spring-mass system. In the ideal case, the gravity force accounts for a large percent of the variance in the movement. But when there are more motions involved, either in the filming process, or on the mass itself, we would need more component to explain the movement. We have also noticed that the filming direction does not impact the result of the analysis, but the motion needs to start at the same point, otherwise there would be a significant lag from our result that renders the results useless.

Appendix A MATLAB Functions

- `implay('file')` opens a movie player, displaying the content of the `file`. It reads one frame at a time, and does not play audio tracks.
- `imshow(I)` shows the gray scale of image `I`.
- `I = rgb2gray(RGB)` converts RGB image `I` to gray scale.
- `I = uint8(X)` converts `X` to unsigned 8-bit integer. The values of a uint8 range from 0 to 255.
- `I = find(X)` find indices of elements based on the logical expression `X`.
- `diag(X)` returns the diagonal of matrix `X`.

Appendix B MATLAB Code

```
clear all; close all; clc

load('cam1_1.mat')
load('cam1_2.mat')
load('cam1_3.mat')
load('cam1_4.mat')
load('cam2_1.mat')
load('cam2_2.mat')
load('cam2_3.mat')
load('cam2_4.mat')
load('cam3_1.mat')
load('cam3_2.mat')
load('cam3_3.mat')
load('cam3_4.mat')
%%
implay(vidFrames1_1)

%% Test 1: Ideal case
numFrames1 = size(vidFrames1_1, 4);
numFrames2 = size(vidFrames2_1, 4);
numFrames3 = size(vidFrames3_1, 4);

x_1 = [];
y_1 = [];

crop = zeros(480,640);
crop(200:430,300:400) = ones(231,101);

for j = 1:numFrames1
    X1 = rgb2gray(vidFrames1_1(:,:,:,j));
    X1 = uint8(crop) .* X1;
    [M, I] = max(double(X1(:)));
    [row, col] = find(X1 >= 0.92 * M);
    x_1(j) = mean(col);
    y_1(j) = mean(row);
end

[~, i] = max(y_1(1:50));
x_1 = x_1 (i:end);
y_1 = y_1 (i:end);

y_2 = [];
x_2 = [];

crop = zeros(480,640);
crop(100:400,250:350) = ones(301,101);

for j = 1:numFrames2
    X2 = rgb2gray(vidFrames2_1(:,:,:,j));
    X2 = uint8(crop) .* X2;
    [M, I] = max(double(X2(:)));
    [row, col] = find(X2 >= 0.92 * M);
```

```
    x_2(j) = mean(col);
    y_2(j) = mean(row);
end

[~, i] = max(y_2(1:50));
x_2 = x_2(i:end);
y_2 = y_2(i:end);

x_3 = [];
y_3 = [];

crop = zeros(480,640);
crop(250:350,250:450) = ones(101,201);

for j = 1:numFrames3
    X3 = rgb2gray(vidFrames3_1(:,:,:,j));
    X3 = uint8(crop) .* X3;
    [M, I] = max(double(X3(:)));
    [row, col] = find(X3 >= 0.92 * M);
    x_3(j) = mean(col);
    y_3(j) = mean(row);
end
[~, i] = max(y_3(1:50));
x_3 = x_3(i:end);
y_3 = y_3(i:end);

minLength = min([length(y_1), length(y_2), length(y_3)]);
X = [x_1(1:minLength); y_1(1:minLength); x_2(1:minLength);
    y_2(1:minLength); x_3(1:minLength); y_3(1:minLength)];
[~, n] = size(X);
avg = mean(X, 2);
X = X - repmat(avg,1,n);
[U, S, V] = svd(X, 'econ');

figure()
subplot(1, 3, 1)
plot(diag(S)./sum(diag(S)), 'k.', 'MarkerSize', 20)
xlabel('Principal Component')
ylabel('Energy')
title('Ideal Case Energy in Each Component')
xlim([1 7])

subplot(1, 3, 2)
plot(V(:,1));
hold on;
plot(V(:,2));
hold off;
xlabel('Time (frames)')
ylabel('Displacement')
title('Ideal Case - Displacement across Principle Component')
legend('PC1', 'PC2')

subplot(1, 3, 3)
V = V * S;
```

```
plot(V(:,1));
hold on;
plot(V(:,2));
plot(V(:,3));
plot(V(:,4));
plot(V(:,5));
plot(V(:,6));
hold off;
xlabel('Time (frames)')
ylabel('Displacement')
title('Ideal Case - Variance across Principle Component')
legend('PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6')

%% Cropping visualization
figure()
subplot(1, 2, 1)
X1 = rgb2gray(vidFrames1_1(:,:,:,50));
imshow(X1);

subplot(1, 2, 2)
crop = zeros(480,640);
crop(200:430,300:400) = ones(231,101);
X1 = uint8(crop) .* X1;
imshow(X1);

%% XY postion plot for the ideal case
figure()
subplot(1, 3, 1)
plot(x_1, y_1,'k.')
set(gca, 'YDir','reverse')
xlim([275 375])
ylim([250 400])
xlabel('X position')
ylabel('Y position')
title('Camera 1')

subplot(1, 3, 2)
plot(x_2,y_2,'k.')
set(gca, 'YDir','reverse')
xlim([250 350])
ylim([150 350])
xlabel('X position')
ylabel('Y position')
title('Camera 2')

subplot(1, 3, 3)
plot(x_3,y_3,'k.')
set(gca, 'YDir','reverse')
xlim([300 500])
ylim([260 310])
xlabel('X position')
ylabel('Y position')
title('Camera 3')
```

```
%% Test 2: Noisy Case
numFrames1 = size(vidFrames1_2, 4);
numFrames2 = size(vidFrames2_2, 4);
numFrames3 = size(vidFrames3_2, 4);

x_1 = [];
y_1 = [];

crop = zeros(480,640);
crop(200:430,250:450) = ones(231,201);

for j = 1:numFrames1
    X1 = rgb2gray(vidFrames1_2(:,:,:,j));
    X1 = uint8(crop) .* X1;
    [M, I] = max(double(X1));
    [~,indx] = max(M);
    indy = I(indx);
    x_1(j) = (indx);
    y_1(j) = (indy);
end

[~,i] = max(y_1(1:50));
x_1 = x_1 (i:end);
y_1 = y_1 (i:end);

x_2 = [];
y_2 = [];

crop = zeros(480,640);
crop(80:430,150:450) = ones(351,301);

for j = 1:numFrames2
    X2 = rgb2gray(vidFrames2_2(:,:,:,j));
    X2 = uint8(crop) .* X2;
    [M, I] = max(double(X2));
    [~,indx] = max(M);
    indy = I(indx);
    x_2(j) = (indx);
    y_2(j) = (indy);
end

[~,i] = max(y_2(1:50));
x_2 = x_2(i:end);
y_2 = y_2(i:end);

x_3 = [];
y_3 = [];

crop = zeros(480,640);
crop(200:430,250:450) = ones(231,201);

for j = 1:numFrames3
    X3 = rgb2gray(vidFrames3_2(:,:,:,j));
    X3 = uint8(crop) .* X3;
```

```
    [M, I] = max(double(X3));
    [~,indx] = max(M);
    indy = I(indx);
    x_3(j) = (indx);
    y_3(j) = (indy);
end

[~,i] = max(y_3(1:50));
x_3 = x_3(i:end);
y_3 = y_3(i:end);

minLength = min([length(y_1), length(y_2), length(y_3)]);
X = [x_1(1:minLength); y_1(1:minLength); x_2(1:minLength);
    y_2(1:minLength); x_3(1:minLength); y_3(1:minLength)];
[~, n] = size(X);
avg = mean(X, 2);
X = X - repmat(avg,1,n);
[~, S, V] = svd(X, 'econ');

figure()
subplot(1, 3, 1)
plot(diag(S)./sum(diag(S)), 'k.', 'MarkerSize', 20)
xlabel('Principal Component')
ylabel('Energy')
title('Noisy Case Energy in Each Component')
xlim([1 7])

subplot(1, 3, 2)
plot(V(:,1));
hold on;
plot(V(:,2));
hold off;
xlabel('Time (frames)')
ylabel('Displacement')
title('Noisy Case - Displacement across Principle Component')
legend('PC1', 'PC2')

subplot(1, 3, 3)
V = V * S;
plot(V(:,1));
hold on;
plot(V(:,2));
plot(V(:,3));
plot(V(:,4));
plot(V(:,5));
plot(V(:,6));
hold off;
xlabel('Time (frames)')
ylabel('Displacement')
title('Noisy Case - Variance across Principle Component')
legend('PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6')

%% XY postion plot for the noisy case
figure()
```

```
subplot(1, 3, 1)
plot(x_1,y_1,'k.')
set(gca, 'YDir','reverse')
xlim([100 600])
ylim([100 500])
xlabel('X position')
ylabel('Y position')
title('Camera 1')

subplot(1, 3, 2)
plot(x_2,y_2,'k.')
set(gca, 'YDir','reverse')
xlim([0 600])
ylim([0 600])
xlabel('X position')
ylabel('Y position')
title('Camera 2')

subplot(1, 3, 3)
plot(x_3,y_3,'k.')
set(gca, 'YDir','reverse')
xlim([100 600])
ylim([150 350])
xlabel('X position')
ylabel('Y position')
title('Camera 3')


%% Test 3: Horizontal Displacement
numFrames1 = size(vidFrames1_3, 4);
numFrames2 = size(vidFrames2_3, 4);
numFrames3 = size(vidFrames3_3, 4);

x_1 = [];
y_1 = [];

crop = zeros(480,640);
crop(200:380,280:400) = ones(181,121);

for j = 1:numFrames1
    X1 = rgb2gray(vidFrames1_3(:,:,:,j));
    X1 = uint8(crop) .* X1;
    [M, I] = max(double(X1));
    [~,indx] = max(M);
    indy = I(indx);
    y_1(j) = (indy);
    x_1(j) = (indx);
end

[~,i] = max(y_1(1:50));
x_1 = x_1 (i:end);
y_1 = y_1 (i:end);

x_2 = [];
```

```
y_2 = [];

crop = zeros(480,640);
crop(170:400,200:400) = ones(231,201);

for j = 1:numFrames2
    X2 = rgb2gray(vidFrames2_3(:,:,:,j));
    X2 = uint8(crop) .* X2;
    [M, I] = max(double(X2));
    [~,indx] = max(M);
    indy = I(indx);
    x_2(j) = (indx);
    y_2(j) = (indy);
end

[~,i] = max(y_2(1:50));
x_2 = x_2(i:end);
y_2 = y_2(i:end);

x_3 = [];
y_3 = [];

crop = zeros(480,640);
crop(200:300,250:450) = ones(101,201);

for j = 1:numFrames3
    X3 = rgb2gray(vidFrames3_3(:,:,:,j));
    X3 = uint8(crop) .* X3;
    [M, I] = max(double(X3));
    [~,indx] = max(M);
    indy = I(indx);
    x_3(j) = (indx);
    y_3(j) = (indy);
end

[~,i] = max(y_3(1:50));
x_3 = x_3(i:end);
y_3 = y_3(i:end);

minLength = min([length(y_1), length(y_2), length(y_3)]);
X = [x_1(1:minLength); y_1(1:minLength); x_2(1:minLength);
    y_2(1:minLength); x_3(1:minLength); y_3(1:minLength)];
[~, n] = size(X);
avg = mean(X, 2);
X = X - repmat(avg,1,n);
[~, S, V] = svd(X, 'econ');

figure()
subplot(1, 3, 1)
plot(diag(S)./sum(diag(S)), 'k.', 'MarkerSize', 20)
xlabel('Principal Component')
ylabel('Energy')
title('Horizontal Case Energy in Each Component')
xlim([1 7])
```

```
subplot(1, 3, 2)
plot(V(:,1));
hold on;
plot(V(:,2));

hold off;
xlabel('Time (frames)')
ylabel('Displacement')
title('Horizontal Case - Displacement across Principle Component')
legend('PC1', 'PC2')

subplot(1, 3, 3)
V = V * S;
plot(V(:,1));
hold on;
plot(V(:,2));
plot(V(:,3));
plot(V(:,4));
plot(V(:,5));
plot(V(:,6));
hold off;
xlabel('Time (frames)')
ylabel('Displacement')
title('Horizontal Case - Variance across Principle Component')
legend('PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6')


%% XY postion plot for the horizontal case
figure()
subplot(1, 3, 1)
plot(x_1,y_1,'k.')
set(gca, 'YDir','reverse')
xlim([100 600])
ylim([100 500])
xlabel('X position')
ylabel('Y position')
title('Camera 1')

subplot(1, 3, 2)
plot(x_2,y_2,'k.')
set(gca, 'YDir','reverse')
xlim([0 600])
ylim([0 600])
xlabel('X position')
ylabel('Y position')
title('Camera 2')

subplot(1, 3, 3)
plot(x_3,y_3,'k.')
set(gca, 'YDir','reverse')
xlim([100 600])
ylim([150 350])
xlabel('X position')
```

```
ylabel('Y position')
title('Camera 3')

%% Test 4: Horizontal Displacement and Rotation
numFrames1 = size(vidFrames1_4, 4);
numFrames2 = size(vidFrames2_4, 4);
numFrames3 = size(vidFrames3_4, 4);

y1 = [];
x1 = [];

crop = zeros(480,640);
crop(200:380,300:500) = ones(181,201);

for j = 1:numFrames1
    X1 = rgb2gray(vidFrames1_4(:,:,:,j));
    X1 = uint8(crop) .* (X1);
    [M, I] = max(double(X1(:)));
    [row, col] = find(X1 >= 0.95*M);
    x_1(j) = mean(col);
    y_1(j) = mean(row);
end

[~,i] = max(y_1(1:50));
x_1 = x_1(i:end);
y_1 = y_1(i:end);

y2 = [];
x2 = [];

crop = zeros(480,640);
crop(100:320,230:430) = ones(221,201);

for j = 1:numFrames2
    X2 = rgb2gray(vidFrames2_4(:,:,:,j));
    X2 = uint8(crop) .* (X2);
    [M, I] = max(double(X2(:)));
    [row,col]=find(X2 >= 0.95*M);
    x_2(j) = mean(col);
    y_2(j) = mean(row);
end

[~,i] = max(y_2(1:50));
x_2 = x_2(i:end);
y_2 = y_2(i:end);

x_3 = [];
y_3 = [];

crop = zeros(480,640);
crop(150:270,250:450) = ones(121,201);

for j = 1:numFrames3
    X3 = (rgb2gray(vidFrames3_4(:,:,:,j)));
```

```
    X3 = uint8(crop) .* (X3);
    [M, I] = max(double(X3(:)));
    [row, col]=find(X3 >= 0.9*M);
    x_3(j) = mean(col);
    y_3(j) = mean(row);
end

[~,i] = max(y_3(1:50));
x_3 = x_3(i:end);
y_3 = y_3(i:end);

minLength = min([length(y_1), length(y_2), length(y_3)]);
X = [x_1(1:minLength); y_1(1:minLength); x_2(1:minLength);
    y_2(1:minLength); x_3(1:minLength); y_3(1:minLength)];
[~, n] = size(X);
avg = mean(X, 2);
X = X - repmat(avg,1,n);
[~, S, V] = svd(X, 'econ');

figure()
subplot(1, 3, 1)
plot(diag(S)./sum(diag(S)), 'k.', 'MarkerSize', 20)
xlabel('Principal Component')
ylabel('Energy')
title('Horizontal Displacement and Rotation Case Energy in Each Component')
xlim([1 7])

subplot(1, 3, 2)
plot(V(:,1));
hold on;
plot(V(:,2));
hold off;
xlabel('Time (frames)')
ylabel('Displacement')
title('Horizontal and Rotation Case - Displacement across Principle Component')
legend('PC1', 'PC2')

subplot(1, 3, 3)
V = V * S;
plot(V(:,1));
hold on;
plot(V(:,2));
plot(V(:,3));
plot(V(:,4));
plot(V(:,5));
plot(V(:,6));
hold off;
xlabel('Time (frames)')
ylabel('Displacement')
title('Horizontal and Rotation Case - Variance across Principle Component')
legend('PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6')

%% XY postion plot for the Horizontal Displacement and Rotation case
figure()
```

```
subplot(1, 3, 1)
plot(x_1,y_1,'k.')
set(gca, 'YDir','reverse')
xlim([100 600])
ylim([100 500])
xlabel('X position')
ylabel('Y position')
title('Camera 1')

subplot(1, 3, 2)
plot(x_2,y_2,'k.')
set(gca, 'YDir','reverse')
xlim([0 600])
ylim([0 600])
xlabel('X position')
ylabel('Y position')
title('Camera 2')

subplot(1, 3, 3)
plot(x_3,y_3,'k.')
set(gca, 'YDir','reverse')
xlim([100 600])
ylim([150 350])
xlabel('X position')
ylabel('Y position')
title('Camera 3')
```