AMATH 482 Homework 3

Jessica Shi, 1762333

February 20, 2021

Abstract

In this homework, we use the Principal Component Analysis to analyze the video of spring-mass systems from different cameras. We use PCA to track the motion of the spring along different axis and planes. We also consider the noises(camera shakes) to compare its result with normal cases.

1 Introduction and Overview

In this homework, I have 12 videos to analyze, which are taken from different cameras and various angles under 4 different conditions. The video is about the spring-mass system. I would use the Principal Component Analysis to analyze the videos and find the motion of the spring. There are 4 different cases. In the ideal case, only simple harmonic motion in the z direction are under consideration. In the noisy case, we introduce the camera shakes as noise in the video. In the horizontal displacement case, the mass is released off-center so there are some horizontal displacement in the xy plane. Finally, in the horizontal displacement and rotation case, besides the off-center release, there are rotations in the xy plane. I would use the PCA to track the motion of the mass.

2 Theoretical Background

The fundamental concept used in this homework is the Principal Component Analysis. And it is a derived method from the Singular Value Decomposition.

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

This is the SVD of a matrix $\mathbf{A}$. The vectors $\mathbf{u}_n$ in $\mathbf{U}$ are the left singular vectors while the vectors $\mathbf{v}_n$ in $\mathbf{V}$ are right singular vectors. And the σ_n on the diagonal of σ are called the singular values of $\mathbf{A}$. $\mathbf{U}$ and $\mathbf{V^*}$ are unitary matrices and σ is a diagonal matrix. Note that $\mathbf{V^*}$ is the transpose of $\mathbf{V}$.The following steps from lecture notes shows how we calculate it.

We know that

$$\begin{aligned} \mathbf{A^*A} &= (\mathbf{U\Sigma V^*})^*(\mathbf{U\Sigma V^*}) \\ &= \mathbf{V\Sigma U^*U\Sigma V^*} \\ &= \mathbf{V\Sigma^2 V^*} \end{aligned} \tag{2}$$

Then, multiply $\mathbf{V}$ to the right, we get

$$\mathbf{A^*AV} = \mathbf{V\Sigma^2} \tag{3}$$

By doing this, the columns of $\mathbf{V}$ are eigenvectors of $\mathbf{A^*A}$ and the eigenvalues are the square of the singular values.

Similarly, we could multiply both sides with $\mathbf{U}$ to solve for $\mathbf{U}$.

$$\mathbf{A*A} = \mathbf{U\Sigma^2 U*} \tag{4}$$

$$\mathbf{A*AU} = \mathbf{U\Sigma^2} \tag{5}$$

I use SVD here for the PCA algorithm. This algorithm is used to calculate the principal components and then perform change of basis of data. It helps us reduce the dimension of data. Here we introduce the covariance matrix. Recall that the SVD of a matrix is connected to the eigenvalue decomposition. Therefore, we consider:

$$\mathbf{A} = \frac{1}{\sqrt{n-1}}\mathbf{X} \tag{6}$$

$$\mathbf{C_X} = \frac{1}{n-1}\mathbf{XX*} = \mathbf{AA*} \tag{7}$$

Also, we know that:

$$\mathbf{C_X} = \mathbf{AA*} = \mathbf{U\Sigma^2 U*} \tag{8}$$

(Note: I use * to represent the transpose.)

So here the eigenvalues of covariance matrix are the squares of the scaled singular values. In the actual implementation, I would use the builtin function `svd(A)` to get the corresponding $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V*}$ matrices for $\mathbf{A}$.

3 Algorithm Implementation and Development

The specific implementation is done using MATLAB. I have 12 videos taken from 3 different cameras under four different experiment settings. Thus, the process of analyzing the 4 cases is similar. Each time I consider three different videos. And finally I would compare the result of each case.

First, I load all the video files into MATLAB, find the size of video (in pixels) and then filter out irrelevant information. I use a filter matrix and try different ranges for different videos. The resulting focus range for different videos are not the same since they are taken from different angles. I implement a function `process_data` to do the filtering of data. The specific range are described in the MATLAB code.

Then I try to track the motion of the mass by track the light in the video. I use the `min` function to find the minimum value and the corresponding index. And we find the video with smallest length and adjust the length of other two videos to make them consistent in size. Then I put them together in one big matrix. I use the function `get_data` to do these processes. Next, I subtract the mean of each row from each single row in the matrix to standardize the matrix. Then I divide the result by $\sqrt{n-1}$. I am now able to do the `svd` of the resulting matrix. As a result, I get the singular values of the matrix and also the principal component projections.

I repeat the above-mentioned steps for all 4 different cases and make the variance, energy plots, the displacement in z-axis, xy-plane as well as the displacement along the principal component projections.

4 Computational Results

4.1 Case 1: Ideal Case

In this case, I found a principal component with 88.89% energy, which consists the major portion of the total energy. And the second component is about 10%, as shown in Figure 1. By looking at Figure 2 and the case description, we know that the mass is just moving in the z-axis direction. Thus, it is reasonable to get one major principal component. By comparing Figure 2(a) with Figure 2(b), we could see that the motion is consistent and the principal component represents the oscillation correctly.

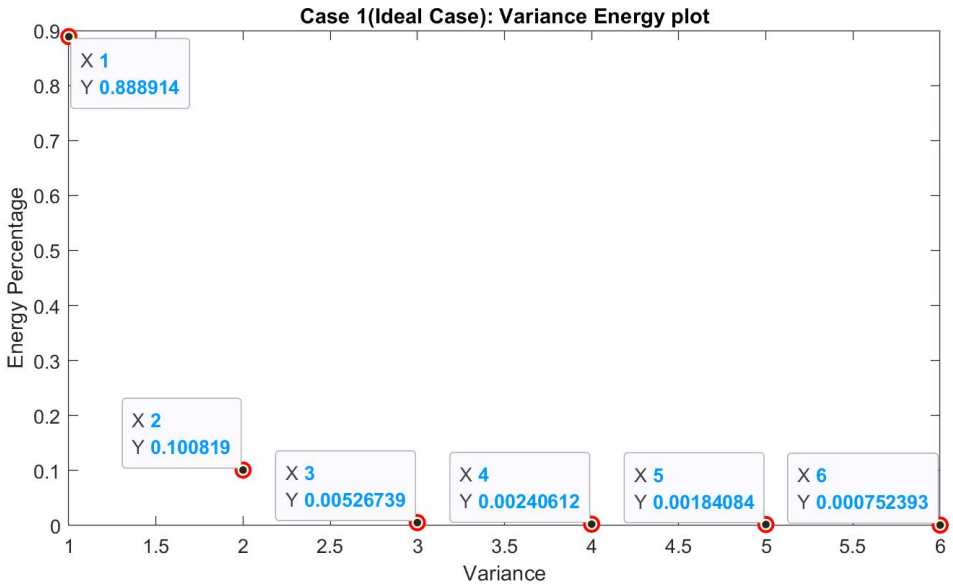
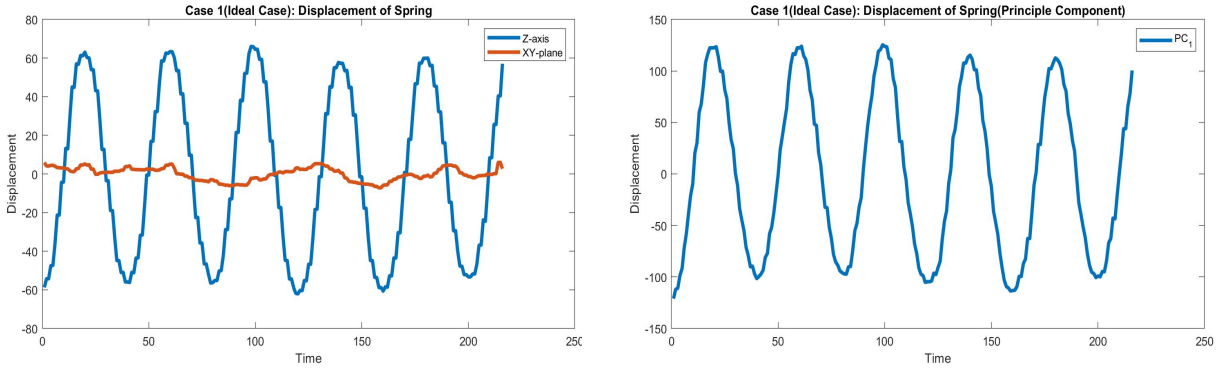


Figure 1: Case 1: Variance and Energy



(a) Case 1: Displacement along z-axis and xy-plane

(b) Case 1: Displacement along Principal Component

Figure 2: Case 1

4.2 Case 2: Noisy Case

In this case, the principal component with most energy is about 64.29% and the second one has about 16.97%, as shown in Figure 3. Due to the experiment settings, the cameras are shaking. Thus it is reasonable that there are some movements along the xy-plane. Therefore, I think that 2 principal components are necessary for this case. We could see from Figure 4 that although the motion is more irregular than the previous case, we could capture a basic oscillation.

4.3 Case 3: Horizontal Displacement

In this case, from Figure 5, we could know that there are 4 components with a non-trivial energy, which are 49.94%, 22.57%, 16.33% and 10.4%. According to the experiment settings, we know that the mass is released off-center, therefore, there are motions along xy-plane. So it is reasonable to consider more principal components in this case. As we could see from Figure 6, the oscillation pattern is clear.

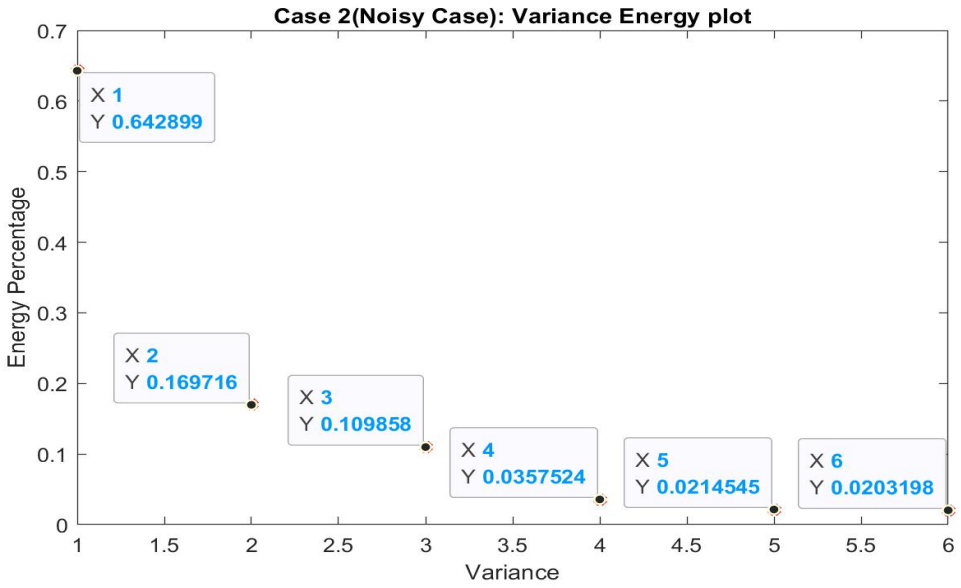
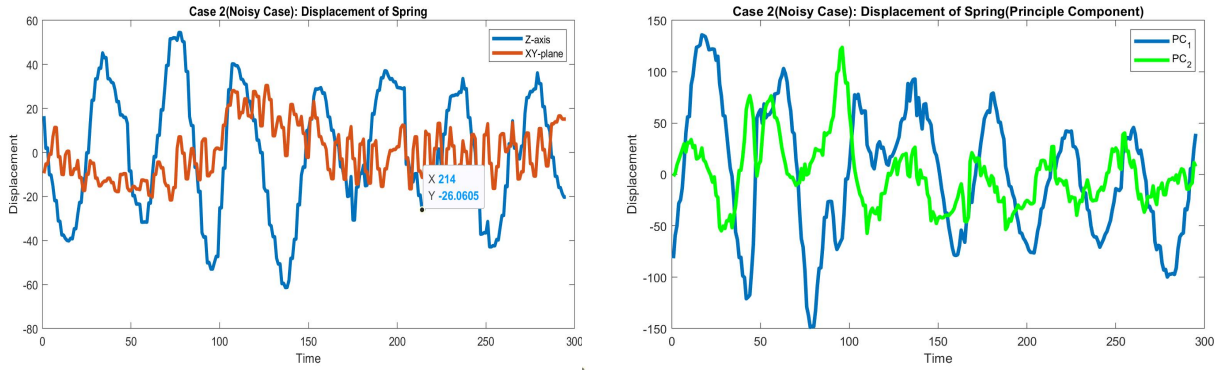


Figure 3: Case 2: Variance and Energy



(a) Case 2: Displacement along z-axis and xy-plane

(b) Case 2: Displacement along Principal Component

Figure 4: Case 2

4.4 Case 4: Horizontal Displacement and Rotation

In this case, the complexity of the experiment increases again. This time, it includes rotation of the video. Now we have motions in z-axis, xy-plane and also rotation to consider. This means that we also need multiple principal components. According to Figure 7, the main components have the energy of 69.11%, 18.02% and 9.04%. I then consider these 3 components. From Figure 8, we could see that the basic oscillation and horizontal movements are captured well.

5 Summary and Conclusions

To conclude, in order to represent the motion of the mass in the spring-mass system videos, we could use the Principal Component Analysis ans the Singular Value Decomposition. After focusing on useful range, we could track the min value and using the corresponding index to do the PCA algorithm. We use svd to calculate the energy percentage to get the proper number of components used in each case. From the resulting figure, it works well. For the first 2 simple cases, fewer principal components are required to show

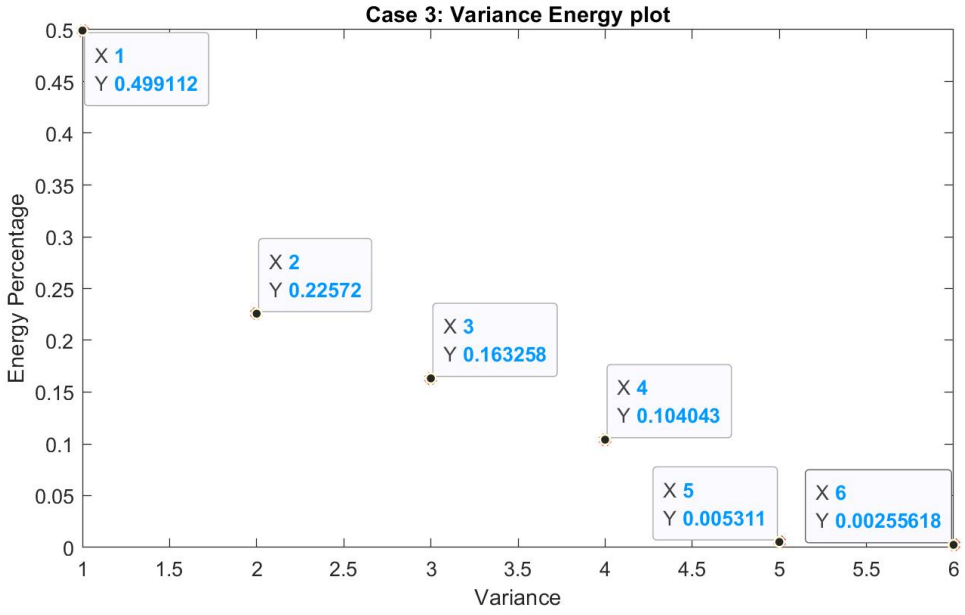


Figure 5: Case 3: Variance and Energy

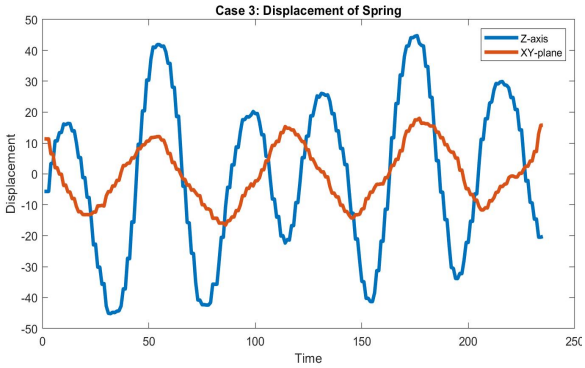
the motion. In the noisy case, however, the motion is influenced largely and the oscillation pattern seems irregular. Fortunately, we could still capture a basic oscillation movement. On the other hand, for the last 2 cases, due to the increase in complexity, the number of principal components needed increases as well in order to captrue the movements.

Appendix A MATLAB Functions

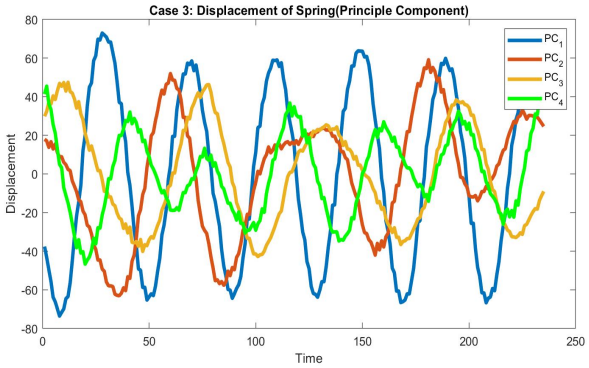
- `I = rgb2gray(RGB)` converts the truecolor image RGB to the grayscale image `I`.
- `x = diag(A)` returns a column vector of the main diagonal elements of `A`.
- `[M,I] = min(A)` returns the index into the operating dimension that corresponds to the minimum value of `A`.
- `plot(X,Y)` creates a 2-D line plot of the data in `Y` versus the corresponding values in `X`.
- `[row,col] = find(__)` returns the row and column subscripts of each nonzero element in array `X` using any of the input arguments in previous syntaxes.
- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix `A`, such that `A = U*S*V'`.

Appendix B MATLAB Code

```
function result = process_data(vidFrame, filter, scale)
    numFrames = size(vidFrame, 4);
    result = zeros(numFrames, 2);
    for i = 1:numFrames
        X = vidFrame(:,:,:,i);
        X_gray = double(rgb2gray(X));
        X_prime = X_gray .* filter;
        threshold = X_prime > scale;
```



(a) Case 3: Displacement along z-axis and xy-plane



(b) Case 3: Displacement along Principal Component

Figure 6: Case 3

```
9
10          [Y, X] = find(threshold);
11          result(i, 1) = mean(X);
12          result(i, 2) = mean(Y);
13      end
14
15  end
```

```
1   function data = get_data(cam1, cam2, cam3)
2       [M, I] = min(cam1(1:20, 2));
3       cam1 = cam1(I:end, :);
4       [M, I] = min(cam2(1:20, 2));
5       cam2 = cam2(I:end, :);
6       [M, I] = min(cam3(1:20, 2));
7       cam3 = cam3(I:end, :);
8
9       % make the 3 cam data consistent in length:
10      len1 = length(cam1);
11      len2 = length(cam2);
12      len3 = length(cam3);
13
14      l = [len1 len2 len3];
15      m = min(l);
16
17      if len1 == m
18          cam2 = cam2(1:m, :);
19          cam3 = cam3(1:m, :);
20      elseif len2 == m
21          cam1 = cam1(1:m, :);
22          cam3 = cam3(1:m, :);
23      else
24          cam1 = cam1(1:m, :);
25          cam2 = cam2(1:m, :);
26      end
27
28      data = [cam1'; cam2'; cam3'];
29  end
```

```
1   % clear all; close all; clc
2
3   % load all the movies
4   load('cam1_1.mat')
5   load('cam1_2.mat')
```

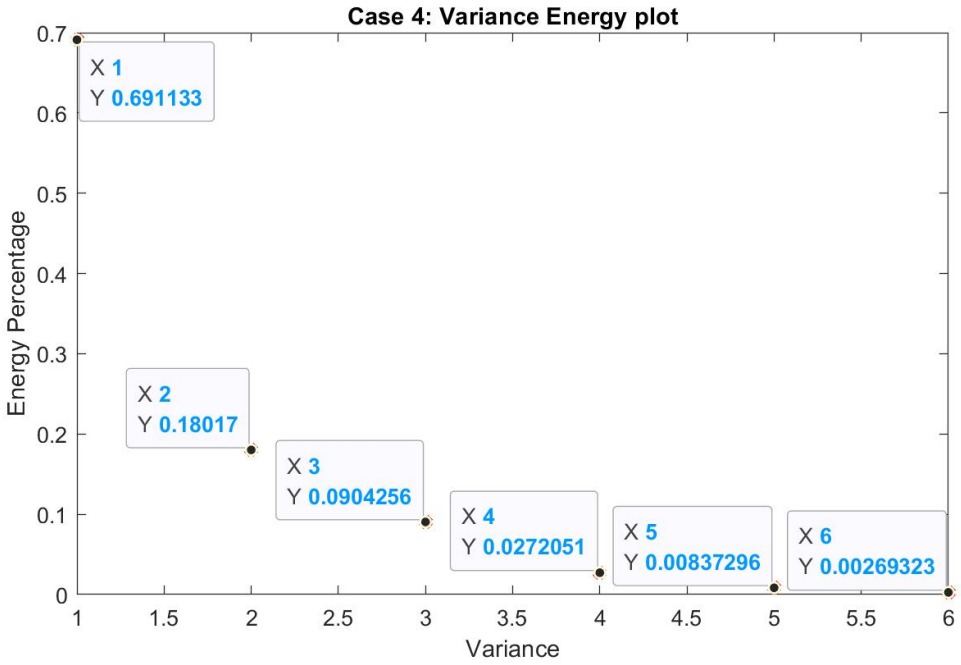
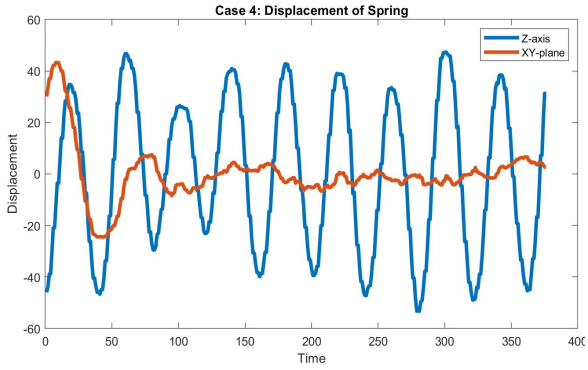
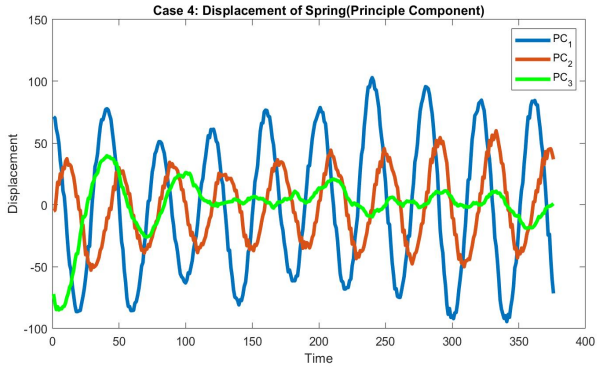


Figure 7: Case 4: Variance and Energy

```
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')


% Test 1: Ideal case ------------------------------------------------------
% size of matrix for cam1_1: 480x640
f1 = zeros(480, 640);
% range of valid data
f1(200:450, 280:400) = 1;
cam1 = process_data(vidFrames1_1, f1, 240);

% size of matrix for cam2_1: 480x640
f2 = zeros(480, 640);
% range of valid data
f2(80:400, 220:350) = 1;
cam2 = process_data(vidFrames2_1, f2, 240);

% size of matrix for cam3_1: 480x640
f3 = zeros(480, 640);
% range of valid data
f3(220:340, 250:500) = 1;
cam3 = process_data(vidFrames3_1, f3, 240);


data = get_data(cam1, cam2, cam3);

% Use SVD
[M, N] = size(data);
```



(a) Case 4: Displacement along z-axis and xy-plane



(b) Case 4: Displacement along Principal Component

Figure 8: Case 4

```
data = data - repmat(mean(data, 2), 1, N);
[U, S, V] = svd(data'/sqrt(N - 1));
sigma = diag(S).^2;
Y = data' * V;

Plot the result
figure(1)
plot(1:6, sigma / sum(sigma), 'ro', 'Linewidth', 3);
title('Case 1(Ideal Case): Variance Energy plot');
xlabel('Variance');
ylabel('Energy Percentage');

figure(2)
plot(1:N, data(2,:), 1:N, data(1,:), 'Linewidth', 3);
xlabel('Time');
ylabel('Displacement');
title('Case 1(Ideal Case): Displacement of Spring');
legend('Z-axis', 'XY-plane');

figure(3)
plot(1:N, Y(:, 1), 'Linewidth', 3);
xlabel('Time');
ylabel('Displacement');
title('Case 1(Ideal Case): Displacement of Spring(Principle Component)');
legend('PC_1');



% Test 2: Noisy case ------------------------------------------------------
% size of matrix for cam1_2: 480x640
f1 = zeros(480, 640);
% range of valid data
f1(200:400, 300:440) = 1;
cam1 = process_data(vidFrames1_2, f1, 240);

% size of matrix for cam2_2: 480x640
f2 = zeros(480, 640);
% range of valid data
f2(50:410, 180:450) = 1;
cam2 = process_data(vidFrames2_2, f2, 240);

% size of matrix for cam3_2: 480x640
f3 = zeros(480, 640);
% range of valid data
f3(180:340, 270:470) = 1;
cam3 = process_data(vidFrames3_2, f3, 240);

```

```
89
90  data = get_data(cam1, cam2, cam3);
91
92  % Use SVD
93  [M, N] = size(data);
94  data = data - repmat(mean(data, 2), 1, N);
95  [U, S, V] = svd(data'/sqrt(N - 1));
96  sigma = diag(S).^2;
97  Y = data' * V;
98
99  % Plot the result
100 figure(4)
101 plot(1:6, sigma / sum(sigma), 'rx', 'Linewidth', 3);
102 title('Case 2(Noisy Case): Variance Energy plot');
103 xlabel('Variance');
104 ylabel('Energy Percentage');
105
106 figure(5)
107 plot(1:N, data(2,:), 1:N, data(1,:), 'Linewidth', 3);
108 xlabel('Time');
109 ylabel('Displacement');
110 title('Case 2(Noisy Case): Displacement of Spring');
111 legend('Z-axis', 'XY-plane');
112
113 figure(6)
114 plot(1:N, Y(:, 1), 1:N, Y(:, 2), 'g', 'Linewidth', 3);
115 xlabel('Time');
116 ylabel('Displacement');
117 title('Case 2(Noisy Case): Displacement of Spring(Principle Component)');
118 legend('PC_1', 'PC_2');
119
120
121
122 % Test 3: Horizontal Displacement ----------------------------------------
123 % size of matrix for cam1_3: 480x640
124 f1 = zeros(480, 640);
125 % range of valid data
126 f1(250:420, 200:390) = 1;
127 cam1 = process_data(vidFrames1_3, f1, 240);
128
129 % size of matrix for cam2_3: 480x640
130 f2 = zeros(480, 640);
131 % range of valid data
132 f2(170:380, 240:400) = 1;
133 cam2 = process_data(vidFrames2_3, f2, 240);
134
135 % size of matrix for cam3_3: 480x640
136 f3 = zeros(480, 640);
137 % range of valid data
138 f3(180:340, 240:480) = 1;
139 cam3 = process_data(vidFrames3_3, f3, 240);
140
141
142 data = get_data(cam1, cam2, cam3);
143
144 % Use SVD
145 [M, N] = size(data);
146 data = data - repmat(mean(data, 2), 1, N);
147 [U, S, V] = svd(data'/sqrt(N - 1));
148 sigma = diag(S).^2;
149 Y = data' * V;
150
151 % Plot the result
152 figure(7)
153 plot(1:6, sigma / sum(sigma), 'rx', 'Linewidth', 3);
154 title('Case 3: Variance Energy plot');
155 xlabel('Variance');
156 ylabel('Energy Percentage');
```

```
157
158 figure(8)
159 plot(1:N, data(2,:), 1:N, data(1,:), 'Linewidth', 3);
160 xlabel('Time');
161 ylabel('Displacement');
162 title('Case 3: Displacement of Spring');
163 legend('Z-axis', 'XY-plane');
164
165 figure(9)
166 plot(1:N, Y(:, 1), 1:N, Y(:, 2), 1:N, Y(:, 3), 1:N, Y(:, 4), 'g', 'Linewidth', 3);
167 xlabel('Time');
168 ylabel('Displacement');
169 title('Case 3: Displacement of Spring(Principle Component)');
170 legend('PC_1', 'PC_2', 'PC_3', 'PC_4');
171
172
173
174
175 % Test 4:Horizontal Displacement and Rotation -----------------------------
176 % size of matrix for cam1_4: 480x640
177 f1 = zeros(480, 640);
178 % range of valid data
179 f1(230:450, 330:460) = 1;
180 cam1 = process_data(vidFrames1_4, f1, 240);
181
182 % size of matrix for cam2_4: 480x640
183 f2 = zeros(480, 640);
184 % range of valid data
185 f2(100:370, 230:420) = 1;
186 cam2 = process_data(vidFrames2_4, f2, 240);
187
188 % size of matrix for cam3_4: 480x640
189 f3 = zeros(480, 640);
190 % range of valid data
191 f3(140:280, 320:500) = 1;
192 cam3 = process_data(vidFrames3_4, f3, 230);
193
194
195 data = get_data(cam1, cam2, cam3);
196
197 % Use SVD
198 [M, N] = size(data);
199 data = data - repmat(mean(data, 2), 1, N);
200 [U, S, V] = svd(data'/sqrt(N - 1));
201 sigma = diag(S).^2;
202 Y = data' * V;
203
204 % Plot the result
205 figure(10)
206 plot(1:6, sigma / sum(sigma), 'rx', 'Linewidth', 3);
207 title('Case 4: Variance Energy plot');
208 xlabel('Variance');
209 ylabel('Energy Percentage');
210
211 figure(11)
212 plot(1:N, data(2,:), 1:N, data(1,:), 'Linewidth', 3);
213 xlabel('Time');
214 ylabel('Displacement');
215 title('Case 4: Displacement of Spring');
216 legend('Z-axis', 'XY-plane');
217
218 figure(12)
219 plot(1:N, Y(:, 1), 1:N, Y(:, 2), 1:N, Y(:, 3), 'g', 'Linewidth', 3);
220 xlabel('Time');
221 ylabel('Displacement');
222 title('Case 4: Displacement of Spring(Principle Component)');
223 legend('PC_1', 'PC_2', 'PC_3');
```