Amath 482 homework 3: Principal Component Analysis

Andy Zhang

February 20, 2020

Abstract

This report demonstrates the application of Principal Component Analysis (PCA) on the Mass-Spring system videos, filmed by cameras from several angles in different cases, so that we can determine if redundancies exist and still capture the whole dynamic after dimension reduction. This can be done by applying Singular Value Decomposition (SVD) to the snapshot matrix of data.

1 Introduction

PCA is an useful technique to extract low-dimension data from the dynamic, but not having to know the governing rules. Thus, it's a practical tool to deal with large amounts of real-world data. By performing PCA, we are able to extract the features and focus of the data and deal with them efficiently.

Specifically in this report, we are going to check the feasibility of dimension reduction of the data from a Spring-Mass system, filmed by 3 cameras from different angles. Besides, 4 different situations are introduced: the ideal case, that the mass generally moves in z-direction; the noisy case that adds camera shakes to the ideal case; the horizontal displacement case that generates both the harmonic motion in z-direction and pendulum motion in x-y plane; and the last case, that adds rotation of the mass to the horizontal displacement case.

2 Theoretical Background

2.1 Singular Value Decomposition (Full SVD)

SVD is an useful tool when performing dimension reduction, by decomposing any matrix into 2 orthogonal/unitary basis and 1 stretching/compressing matrix. It takes the form of:

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

where the * denotes the (conjugate) transpose of the basis V.

For a $m*n$ size matrix A, both basis matrices U and V are unitary/orthogonal, with a size of $m*m$ and $n*n$ respectively. Σ is a $m*n$ diagonal matrix with non-negative diagonal entries known as singular values, sorted from the largest to the smallest. Basically we can understand it as a basis being stretched/compressed and then being applied(rotated) due to another basis.

2.2 Principal Component Analysis (PCA)

As described in **Introduction**, PCA is often applied for dimension reduction of data, which knowing the governing rules is unnecessary. First, we rearrange the x and y coordinates of the mass generated from each video into a matrix:

$$X = \begin{bmatrix} x_a \\ y_a \\ x_b \\ y_b \\ x_c \\ y_c \end{bmatrix} \tag{2}$$

where the number of columns denote the numbers of data points collected over time (in this case, the number of frames), and the number of rows represent the times of measurements.

We can further construct the covariance matrix C_X showing the variances between every possible pair of data by:

$$C_X = \frac{1}{n-1} X X^T \tag{3}$$

where $\frac{1}{n-1}$ is for normalization. The covariance matrix C_X is a square and symmetric matrix whose diagonal entries represent the variances of particular measurements and the off-diagonals denote the covariances between different measurements. Larger off-diagonals entries imply redundancy in our data.

In order to get an ideal basis that all redundancies have been removed and the largest variances of particular measurements are ordered, the idea of SVD is applied. We first do SVD to X and create a new basis:

$$Y = U * X \tag{4}$$

that represent the principal components projection. And for the variance of Y:

$$C_Y = \frac{1}{n-1} Y Y^T = \frac{1}{n-1} \Sigma^2 \tag{5}$$

Thus, an ideal basis is constructed with only diagonal entries in the covariance matrix, that is, all redundancies between each data pairs are removed.

3 Algorithm Implementation and Development

Prepare data

To start with each case, I load the mat files of videos from three angles. Then I will call $[X(N), Y(N), \tilde{}, frameNumN(N)] = size(vidFramesN_N)$ to find the sizes of videos, which are all 480 * 640, and the corresponding number of frames. We insert a ˜ at Z values because it always generates 3 due to RGB scale of the videos. Since we will transfer the video to gray scale later, we don't need those Z values. Then I watch each video and determine the approximate range of the paint can's movement.

Trace the flashlight in each frame

For each frame of each video, I convert it from RGB to gray scale and the pixels to double precisions, which the largest value 255 represents white and smallest value 0 means black. Since the flashlight is the brightest point most of the time, I will trace it to get the position of the paint can in each frame. By observing each video and applying *pcolor*, we can get the movement interval of the paint can and then set what's outside of that interval to black (0 value), because the whiteboard and reflect light in the background will hinder us from locating the flashlight precisely.

Then, I will get the max value among all pixels in each frame and find a vector of coordinates that are larger than certain **threshold value** multiplies the max value. Averaging this vector gives a much more accurate x and y coordinate for each frame, since it can also average out outliers. Note that the x and y values from camera 3 have to be swapped because it's filming in a reverse angle.

Organize the data

Since each video has different numbers of frames, I first rearrange each vector of coordinates so that the can starts from the highest point. Then I find the minimum length of vector among all data and cut off extras that exceed this length. This is an appropriate approach because the mass is in harmonic motion (and pendulum motion), and getting rid of extra data won't affect the pattern. I then plot the horizontal and vertical positions of the mass against each frame to test for different **threshold values** until the plot shows harmonic motions of the mass in all three angles. After arranging the data to a snapshot matrix **Mx**, I subtract mean from each row of data for normalization.

Perform PCA on snapshot matrix

I then perform SVD on the snapshot matrix **Mx** divided by $\sqrt{n-1}$ for normalization, with matlab build-in command $[U, S, V] = svd(Mx,' econ')$, and construct the principal component matrix Y by multiplying U with $\mathbf{Mx}$. Following then I plot Y against each frame so that I can get the normalized position controlled by each mode and check the feasibility of dimension reduction. The energy carried by each mode (totally 6) can also be found by dividing the corresponding singular value squared from the sum of all singular values squared.

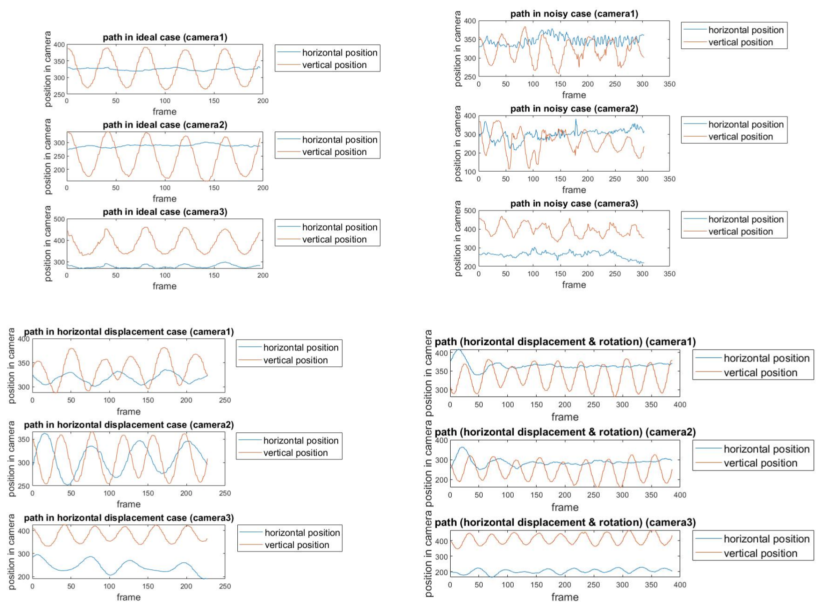

Figure 1: The actual horizontal and vertical positions of the paint can in all videos

4 Computational Results

The horizontal and vertical positions of the paint can from all videos

In **Figure 1**, the actual positions of all cases from all angles are shown. We can see that the mass is in vertical harmonic motion in all cases (except in noisy case) and in horizontal motions in case 3 and 4.

Results from ideal case

In the ideal case shown in **Figure 2**, we can clearly see that the first mode is strongly dominant, which takes more than 95% of the whole energy of the data. Thus, it's feasible and necessary for dimension reduction since we can

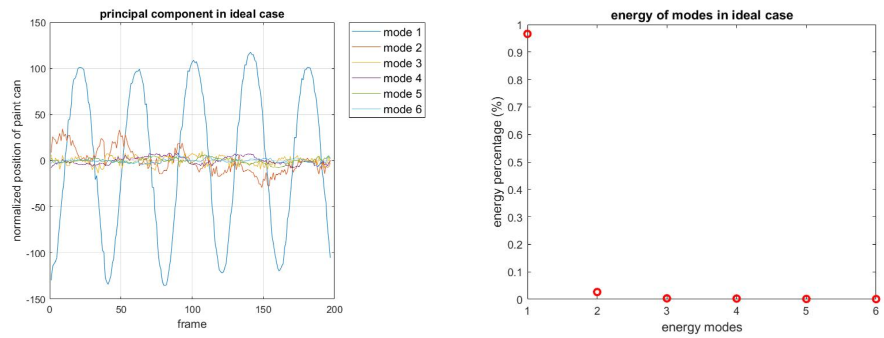

Figure 2: PCA and mode energy in ideal case

also read from the left image that in general, the can is moving only in one direction(z-direction).

Results from noisy case

Shown in **Figure 3**, the dominant mode only takes about 55% of the whole energy due to camera shakes, and four modes are needed to capture 95% of the whole dynamics. Thus, dimension reduction is not necessary because even the last two modes capture about 5% of the dynamics. In this case, PCA is not performing well as we can read from the left image of position.

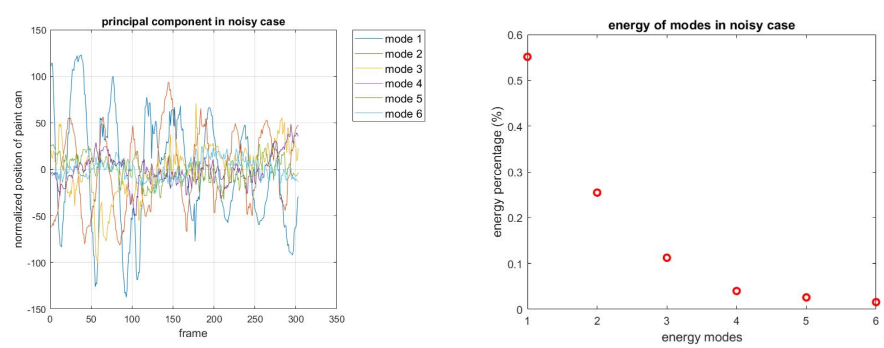

Figure 3: PCA and mode energy in noisy case

Results from horizontal displacement case

In the horizontal displacement case, the paint can is not only moving in z-direction, but also in x-y plane. Thus, data from 2 different angles are necessary to capture both motions and what's shown in **Figure 4**, that 4 modes take approximately all energy in the dynamics makes sense. We can also see from

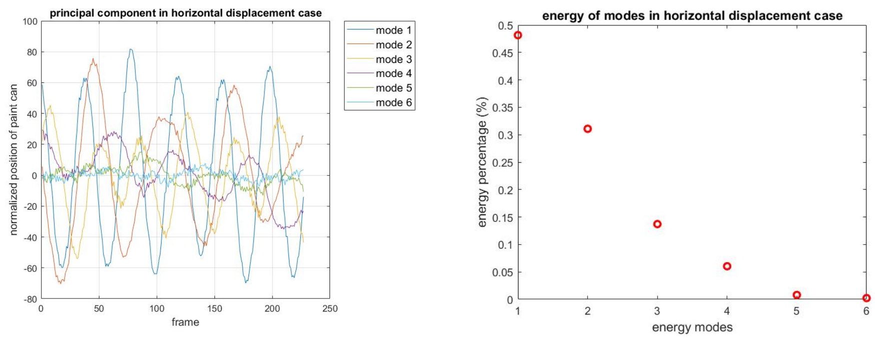

Figure 4: PCA and mode energy in horizontal displacement case

the left image that mode 5 and 6 result in small variations in positions. So we can do dimension reduction in this case so that we capture the whole dynamic by using only two cameras.

Results from horizontal displacement and rotation case

The last case is the horizontal displacement and rotation of the paint can. However, the can is in pendulum motion only in the first few frames (as shown in **Figure 1**). In the rest of the videos, the can is just in simple vertical harmonic motions. What the results shown in **Figure 5**, is that the first dominant mode takes 65% of the energy and only three modes are needed to represent the whole system. However, we are failing to capture the rotation of can due to fixed angles (linearity). So with that being said, we can do dimension reduction in this case but can only represent the pendulum and harmonic motions by applying a half of the data.

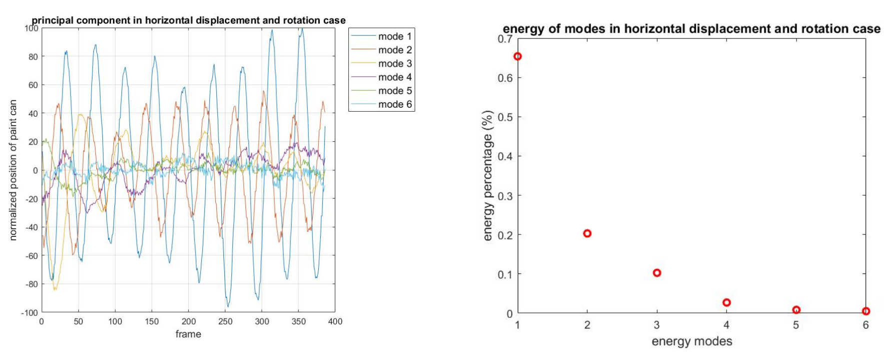

Figure 5: PCA and mode energy in horizontal displacement and rotation case

5 Summary and Conclusion

Through the application of PCA on the data in different cases, we can often extract the dominant components and check redundancy, like in the first and third cases. While in the noisy case, the camera shakes interfere with the performance of the PCA, and dimension reduction might be performed in this case only on the premise of applying methods that remove camera shakes; while in the rotation case, the largest singular value, which represents 65% of the dynamics is quite misleading due to the linearity of SVD methods, and thus, we fail to capture rotation through PCA.

6 Appendix A

double(X) returns the double precision of X.

rgb2gray(X) converts RGB image X to grayscale.

find(M > 0.95 * max) finds all the pixels' positions with values that are larger than 95% of the max values of all pixels.

mean(Mx, 2) returns the mean values of Mx along 2 dimensions, that is, along each row of **Mx**.

repmat(mu, 1, n) creates a matrix with 1 * n copies of **mu**.

[U, S, V] = svd(Mx, 'econ') produces the economy size of SVD, which is efficient for a m-by-n matrix **Mx** that m < n.

7 Appendix B

```
clear; close all; clc;

% Case 1 - ideal

load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')


[X1(1), Y1(1), ~, frameNum1(1)] = size(vidFrames1_1);
[X1(2), Y1(2), ~, frameNum1(2)] = size(vidFrames2_1);
[X1(3), Y1(3), ~, frameNum1(3)] = size(vidFrames3_1);

%{
numFrames = size(vidFrames3_1,4);
for j = 1:numFrames
    X = vidFrames3_1(:,:,:,j);
    imshow(X); drawnow
end
%}
%%
% trace the flashlight for each frame for each camera angle
for j = 1 : frameNum1(1)
    M1 = double(rgb2gray(vidFrames1_1(:, :, :, j)));
    M1(:, [1:300, 400:end]) = 0;
    maxVal = max(M1(:));
    [yval, xval] = find(M1 >= 0.95*maxVal);
    x1(j, 1) = mean(xval);
    y1(j, 1) = mean(yval);
    %{
    pcolor(M1)
    shading interp
    drawnow
    colormap(gray)
    %}
end

for j = 1 : frameNum1(2)
    M2 = double(rgb2gray(vidFrames2_1(:, :, :, j)));
    M2(:, [1:200, 400:end]) = 0;
    maxVal = max(M2(:));
    [yval, xval] = find(M2 >= 0.95*maxVal);
```

```
    x2(j, 1) = mean(xval);
    y2(j, 1) = mean(yval);
    %{
    pcolor(M2)
    shading interp
    drawnow
    colormap(gray)
    %}
end

% rotate back by swapping xval and yval
for j = 1 : frameNum1(3)
    M3 = double(rgb2gray(vidFrames3_1(:, :, :, j)));
    M3(:, [1:240, 480:end]) = 0;
    M3([1:200, 350:end], :) = 0;
    maxVal = max(M3(:));
    [yval, xval] = find(M3 >= 0.95*maxVal);
    x3(j, 1) = mean(yval);
    y3(j, 1) = mean(xval);
    %{
    pcolor(M3)
    shading interp
    drawnow
    colormap(gray)
    %}
end

% cut each data so that the can
% starts from the highest point
x1 = x1(30 : end);
y1 = y1(30 : end);
x2 = x2(39 : end);
y2 = y2(39 : end);
x3 = x3(30 : end);
y3 = y3(30 : end);

% organize data with same length
minlength = min([length(y1), length(y2), length(y3)]);
x1 = x1(1 : minlength);
y1 = y1(1 : minlength);
x2 = x2(1 : minlength);
y2 = y2(1 : minlength);
x3 = x3(1 : minlength);
y3 = y3(1 : minlength);

Mx = [x1, y1, x2, y2, x3, y3];
```

```matlab
Mx = Mx';
frame = 1:minlength;

% examine for the path of paint can and appropriate
% threshold values
figure(1)
for i = 1 : 3
    subplot(3, 1, i)
    plot(frame, Mx(2 * i - 1, :), frame, Mx(2 * i, :))
    legend('horizontal position', 'vertical position', 'FontSize', 12, 'Location', 'bestoutside')
    xlabel('frame', 'FontSize', 12)
    ylabel('position in camera', 'FontSize', 12)
    title(['path in ideal case (camera',num2str(i),')'], 'FontSize', 12)
end

% subtract the mean
[m, n] = size(Mx);
mu = mean(Mx, 2);
Mx = Mx - repmat(mu, 1, n);


% perform PCA
[U, S, V] = svd(Mx / sqrt(n-1), 'econ');
Y = U' * Mx;

figure(2)
plot(frame, Y)
xlabel('frame', 'FontSize', 12)
ylabel('normalized position of paint can', 'FontSize', 12)
title('principal component in ideal case', 'FontSize', 12)
legend('mode 1', 'mode 2', 'mode 3', 'mode 4', 'mode 5', 'mode 6', 'FontSize', 12, 'Location', 'bestoutside')
grid on

figure(3)
sig = diag(S);
plot(sig.^2/sum(sig.^2),'ro','Linewidth',2)
xticks([0:1:6])
xlabel('energy modes', 'FontSize', 12)
ylabel('energy percentage (%)', 'FontSize', 12)
title('energy of modes in ideal case', 'FontSize', 12)
```

```
%% Case 2 - noisy
clear; close all; clc;

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

[X2(1), Y2(1), ~, frameNum2(1)] = size(vidFrames1_2);
[X2(2), Y2(2), ~, frameNum2(2)] = size(vidFrames2_2);
[X2(3), Y2(3), ~, frameNum2(3)] = size(vidFrames3_2);
%{
numFrames = size(vidFrames3_2,4);
for j = 1:numFrames
    X = vidFrames3_2(:,:,:,j);
    imshow(X); drawnow
end
%}
%%
% trace the flashlight for each frame for each camera angle
for j = 1 : frameNum2(1)
    M1 = double(rgb2gray(vidFrames1_2(:, :, :, j)));
    M1(:, [1:300, 400:end]) = 0;
    M1(1:200, :) = 0;
    maxVal = max(M1(:));
    [yval, xval] = find(M1 >= 0.95*maxVal);
    x1(j, 1) = mean(xval);
    y1(j, 1) = mean(yval);
    %{
    pcolor(M1)
    shading interp
    drawnow
    colormap(gray)
    %}
end

for j = 1 : frameNum2(2)
    M2 = double(rgb2gray(vidFrames2_2(:, :, :, j)));
    M2(:, [1:160, 480:end]) = 0;
    M2(400:end, :) = 0;
    maxVal = max(M2(:));
    [yval, xval] = find(M2 >= 0.96*maxVal);
    x2(j, 1) = mean(xval);
    y2(j, 1) = mean(yval);
    %{
```

```
    pcolor(M2)
    shading interp
    drawnow
    colormap(gray)
    %}
end

% rotate back by swopping xval and yval
for j = 1 : frameNum2(3)
    M3 = double(rgb2gray(vidFrames3_2(:, :, :, j)));
    M3(:, [1:260, 500:end]) = 0;
    M3([1:160, 360:end], :) = 0;
    maxVal = max(M3(:));
    [yval, xval] = find(M3 >= 0.93*maxVal);
    x3(j, 1) = mean(yval);
    y3(j, 1) = mean(xval);
    %{
    pcolor(M3)
    shading interp
    drawnow
    colormap(gray)
    %}
end

% cut each data so that the can
% starts from the highest point
x1 = x1(12 : end);
y1 = y1(12 : end);
x2 = x2(5 : end);
y2 = y2(5 : end);
x3 = x3(15 : end);
y3 = y3(15 : end);

% organize data with same length
minlength = min([length(y1), length(y2), length(y3)]);
x1 = x1(1 : minlength);
y1 = y1(1 : minlength);
x2 = x2(1 : minlength);
y2 = y2(1 : minlength);
x3 = x3(1 : minlength);
y3 = y3(1 : minlength);

Mx = [x1, y1, x2, y2, x3, y3];
Mx = Mx';
frame = 1:minlength;

```

```
% examine for the path of paint can and appropriate
% threshold values
figure(1)
for i = 1 : 3
    subplot(3, 1, i)
    plot(frame, Mx(2 * i - 1, :), frame, Mx(2 * i, :))
    legend('horizontal position', 'vertical position', '
        FontSize', 12, 'Location', 'bestoutside')
    xlabel('frame', 'FontSize', 12)
    ylabel('position in camera', 'FontSize', 12)
    title(['path in noisy case (camera',num2str(i),')'],
        'FontSize', 12)
end

% subtract the mean
[m, n] = size(Mx);
mu = mean(Mx, 2);
Mx = Mx - repmat(mu, 1, n);


% perform PCA
[U, S, V] = svd(Mx / sqrt(n-1), 'econ');
Y = U' * Mx;

figure(2)
plot(frame, Y)
xlabel('frame', 'FontSize', 12)
ylabel('normalized position of paint can', 'FontSize',
    12)
title('principal component in noisy case', 'FontSize',
    12)
legend('mode 1', 'mode 2', 'mode 3', 'mode 4', 'mode 5',
    'mode 6', 'FontSize', 12, 'Location', 'bestoutside')
grid on

figure(3)
sig = diag(S);
plot(sig.^2/sum(sig.^2),'ro','Linewidth',2)
xticks([0:1:6])
xlabel('energy modes', 'FontSize', 12)
ylabel('energy percentage (%)', 'FontSize', 12)
title('energy of modes in noisy case', 'FontSize', 12)


%%
clear; close all; clc;
```

```
% Case 3 - horizontal displacement
load('cam1_3.mat')
load('cam2_3.mat')
load('cam3_3.mat')


[X3(1), Y3(1), ~, frameNum3(1)] = size(vidFrames1_3);
[X3(2), Y3(2), ~, frameNum3(2)] = size(vidFrames2_3);
[X3(3), Y3(3), ~, frameNum3(3)] = size(vidFrames3_3);

%%
numFrames = size(vidFrames1_3,4);
for j = 1:numFrames
    X = vidFrames1_3(:,:,:,j);
    imshow(X); drawnow
end
%%
% trace the flashlight for each frame for each camera angle
for j = 1 : frameNum3(1)
    M1 = double(rgb2gray(vidFrames1_3(:, :, :, j)));
    M1(:, [1:270, 400:end]) = 0;
    M1(1:230, :) = 0;
    maxVal = max(M1(:));
    [yval, xval] = find(M1 >= 0.95*maxVal);
    x1(j, 1) = mean(xval);
    y1(j, 1) = mean(yval);
    %{
    pcolor(M1)
    shading interp
    drawnow
    colormap(gray)
    %}
end

for j = 1 : frameNum3(2)
    M2 = double(rgb2gray(vidFrames2_3(:, :, :, j)));
    M2(:, [1:200, 440:end]) = 0;
    M2(1:120, :) = 0;
    maxVal = max(M2(:));
    [yval, xval] = find(M2 >= 0.98*maxVal);
    x2(j, 1) = mean(xval);
    y2(j, 1) = mean(yval);
    %{
    pcolor(M2)
```

```
    shading interp
    drawnow
    colormap(gray)
    %}
end

% rotate back by swopping xval and yval
for j = 1 : frameNum3(3)
    M3 = double(rgb2gray(vidFrames3_3(:, :, :, j)));
    M3(:, [1:200, 480:end]) = 0;
    M3([1:120, 360:end], :) = 0;
    maxVal = max(M3(:));
    [yval, xval] = find(M3 >= 0.95*maxVal);
    x3(j, 1) = mean(yval);
    y3(j, 1) = mean(xval);
    %{
    pcolor(M3)
    shading interp
    drawnow
    colormap(gray)
    %}
end

% cut each data so that the can
% starts from the highest point
x1 = x1(9 : end);
y1 = y1(9 : end);
x2 = x2(9 : end);
y2 = y2(9 : end);
x3 = x3(11 : end);
y3 = y3(11 : end);

% organize data with same length
minlength = min([length(y1), length(y2), length(y3)]);
x1 = x1(1 : minlength);
y1 = y1(1 : minlength);
x2 = x2(1 : minlength);
y2 = y2(1 : minlength);
x3 = x3(1 : minlength);
y3 = y3(1 : minlength);

frame = 1:minlength;
Mx = [x1, y1, x2, y2, x3, y3];
Mx = Mx';

% examine for the path of paint can and appropriate
```

```
% threshold values
figure(1)
for i = 1 : 3
    subplot(3, 1, i)
    plot(frame, Mx(2 * i - 1, :), frame, Mx(2 * i, :))
    legend('horizontal position', 'vertical position', 'FontSize', 12, 'Location', 'bestoutside')
    xlabel('frame', 'FontSize', 12)
    ylabel('position in camera', 'FontSize', 12)
    title(['path in horizontal displacement case (camera',num2str(i),')'], 'FontSize', 12)
end

% subtract the mean
[m, n] = size(Mx);
mu = mean(Mx, 2);
Mx = Mx - repmat(mu, 1, n);


% perform PCA
[U, S, V] = svd(Mx / sqrt(n-1), 'econ');
Y = U' * Mx;

figure(2)
plot(frame, Y)
xlabel('frame', 'FontSize', 12)
ylabel('normalized position of paint can', 'FontSize', 12)
title('principal component in horizontal displacement case', 'FontSize', 12)
legend('mode 1', 'mode 2', 'mode 3', 'mode 4', 'mode 5', 'mode 6', 'FontSize', 12, 'Location', 'bestoutside')
grid on

figure(3)
sig = diag(S);
plot(sig.^2/sum(sig.^2),'ro','Linewidth',2)
xticks([0:1:6])
xlabel('energy modes', 'FontSize', 12)
ylabel('energy percentage (%)', 'FontSize', 12)
title('energy of modes in horizontal displacement case', 'FontSize', 12)

%%
clear; close all; clc;

```

```
% Case 4 - horizontal displacement and rotation
load('cam1_4.mat')
load('cam2_4.mat')
load('cam3_4.mat')


[X4(1), Y4(1), ~, frameNum4(1)] = size(vidFrames1_4);
[X4(2), Y4(2), ~, frameNum4(2)] = size(vidFrames2_4);
[X4(3), Y4(3), ~, frameNum4(3)] = size(vidFrames3_4);

%%
numFrames = size(vidFrames2_4,4);
for j = 1:numFrames
    X = vidFrames2_4(:,:,:,j);
    imshow(X); drawnow
end
%%
% trace the flashlight for each frame for each camera
    angle
for j = 1 : frameNum4(1)
    M1 = double(rgb2gray(vidFrames1_4(:, :, :, j)));
    M1(:, [1:300, 480:end]) = 0;
    M1([1:200, 420:end], :) = 0;
    maxVal = max(M1(:));
    [yval, xval] = find(M1 >= 0.95*maxVal);
    x1(j, 1) = mean(xval);
    y1(j, 1) = mean(yval);
    %{
    pcolor(M1)
    shading interp
    drawnow
    colormap(gray)
    %}
end

for j = 1 : frameNum4(2)
    M2 = double(rgb2gray(vidFrames2_4(:, :, :, j)));
    M2(:, [1:210, 420:end]) = 0;
    M2([1:60, 400:end], :) = 0;
    maxVal = max(M2(:));
    [yval, xval] = find(M2 >= 0.98*maxVal);
    x2(j, 1) = mean(xval);
    y2(j, 1) = mean(yval);
    %{
    pcolor(M2)
    shading interp
```

```
    drawnow
    colormap(gray)
    %}
end

% rotate back by swopping xval and yval
for j = 1 : frameNum4(3)
    M3 = double(rgb2gray(vidFrames3_4(:, :, :, j)));
    M3(:, [1:300, 480:end]) = 0;
    M3([1:120, 320:end], :) = 0;
    maxVal = max(M3(:));
    [yval, xval] = find(M3 >= 0.92*maxVal);
    x3(j, 1) = mean(yval);
    y3(j, 1) = mean(xval);
    %{
    pcolor(M3)
    shading interp
    drawnow
    colormap(gray)
    %}
end

% cut each data so that the can
% starts from the highest point
x1 = x1(7 : end);
y1 = y1(7 : end);
x2 = x2(5 : end);
y2 = y2(5 : end);
x3 = x3(1 : end);
y3 = y3(1 : end);

% organize data with same length
minlength = min([length(y1), length(y2), length(y3)]);
x1 = x1(1 : minlength);
y1 = y1(1 : minlength);
x2 = x2(1 : minlength);
y2 = y2(1 : minlength);
x3 = x3(1 : minlength);
y3 = y3(1 : minlength);

frame = 1:minlength;
Mx = [x1, y1, x2, y2, x3, y3];
Mx = Mx';

% examine for the path of paint can and appropriate
% threshold values
```

```matlab
figure(1)
for i = 1 : 3
    subplot(3, 1, i)
    plot(frame, Mx(2 * i - 1, :), frame, Mx(2 * i, :))
    legend('horizontal position', 'vertical position', 'FontSize', 12, 'Location', 'bestoutside')
    xlabel('frame', 'FontSize', 12)
    ylabel('position in camera', 'FontSize', 12)
    title(['path (horizontal displacement & rotation) (camera',num2str(i),')'], 'FontSize', 12)
end

% subtract the mean
[m, n] = size(Mx);
mu = mean(Mx, 2);
Mx = Mx - repmat(mu, 1, n);

% perform PCA
[U, S, V] = svd(Mx / sqrt(n-1), 'econ');
Y = U' * Mx;

figure(2)
plot(frame, Y)
xlabel('frame', 'FontSize', 12)
ylabel('normalized position of paint can', 'FontSize', 12)
title('principal component in horizontal displacement and rotation case', 'FontSize', 12)
legend('mode 1', 'mode 2', 'mode 3', 'mode 4', 'mode 5', 'mode 6', 'FontSize', 12, 'Location', 'bestoutside')
grid on

figure(3)
sig = diag(S);
plot(sig.^2/sum(sig.^2),'ro','Linewidth',2)
xticks([0:1:6])
xlabel('energy modes', 'FontSize', 12)
ylabel('energy percentage (%)', 'FontSize', 12)
title('energy of modes in horizontal displacement and rotation case', 'FontSize', 12)
```