Amath482 Homework3

Kexin Jiao

Abstract:

The homework mainly focuses on the practice of the Principal Component Analysis and the Singular Value Decomposition(SVD) to solve the actual problem: analyzing datasets recorded by different angled cameras of a spring-mass system under ideal, noisy, horizontal displacement, horizontal displacement and rotation conditions, comparing and contrasting the different cases.

Introduction and overview:

The homework is about analyzing movie files created from 3 different cameras(different angles) of a spring-mass system under the following four conditions, and we are asked to compare and contrast the different cases.

1. Test 1: Ideal Case. Consider a small displacement of the mass in the z direction and the ensuing oscillations. In this case, the entire motion is in the z direction with simple harmonic motion being observed.
2. Test 2: Noisy Case. Repeat the ideal case experiment, but this time introduce camera shakes into the video recording. This should make it more difficult to extract the simple harmonic motion. But, if the shake isn't too bad, the dynamics will still be extracted with the PCA algorithms.
3. Test 3: Horizontal Displacement. In this case the mass is released off-center so as to produce motion in the x-y plane as well as the z direction.
4. Test 4: Horizontal Displacement and Rotation. In this case the mass is released off-center and rotates so as to produce rotation in the x-y plane and motion in the z direction.

Theoretical background:

The Singular Value Decomposition

$A = U\Sigma V^*$ gives a decomposition of the matrix A through the unitary matrices U and V and a diagonal matrix Σ. The U and V matrix are rotation matrices while the Σ is the stretching matrix. The values σ on the diagonal of Σ are singular values; the vectors u(columns of U) are left singular vectors; the vectors v(columns of V) are right singular vectors. When handling non-square matrices of arbitrary size, m*n matrix(m>=n), m-n columns that are orthogonal are added to matrix U to make the decomposition work(U becomes square matrix). m-n rows of zeros is added to the matrix Σ as well. However, the Matlab has a built-in function [U,S,V] = svd(A,'econ') to do the singular value decomposition and produces an economy-size decomposition of m*n matrix A.

Computing the SVD

$$A^*A = (U\Sigma V^*) * (U\Sigma V^*)$$
$$= V\Sigma U^*U\Sigma V^* = V\Sigma^2 V^*$$

Multiplying on the right by V gives $A^*AV = V\Sigma^2$ so that the columns of V are the eigenvectors of A^*A with eigenvalues given by the square of the singular values. Multiplying on the right by U gives $A^*AU = U\Sigma^2$ to solve U.

Principal Component Analysis

$$C_x = 1/(n-1) * XX^T = \begin{bmatrix} \sigma_a^2 & \sigma_{ab}^2 & \sigma_{ac}^2 & \sigma_{ad}^2 \\ \sigma_{ba}^2 & \sigma_b^2 & \sigma_{bc}^2 & \sigma_{bd}^2 \\ \sigma_{ca}^2 & \sigma_{cb}^2 & \sigma_c^2 & \sigma_{cd}^2 \\ \sigma_{da}^2 & \sigma_{db}^2 & \sigma_{dc}^2 & \sigma_d^2 \end{bmatrix}$$

is the covariance matrix of $x = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$

The goal of PCA is to find a new set of coordinates, making the variables uncorrelated. Then, each variable contains completely new information without redundancies. Also, we can tell variables with the largest variance, which contain the most important information about our data. To diagonalize the matrix so that all off-diagonal elements (co-variances) are zero:

$$C_x = V\Lambda V^{-1}$$

Since C_x is a symmetric matrix, it have real eigenvalues and orthogonal corresponding eigenvectors. The basis of eigenvectors contained in V are called the principal components which are uncorrelated since they are orthogonal. The diagonal entries of Λ, the eigenvalues of C_x, are the variances of these new variables.

Connection to SVD

To account for the 1/(n -1) factor, $A = X/\sqrt{n-1}$

Then, $C_x = 1/(n-1) * XX^T = AA^T = U\Sigma^2 U^T$

U is the orthogonal matrix of left-singular vectors and Σ is the diagonal matrix of singular values. So, the eigenvalues of the covariance matrix are the squares of the (scaled) singular values, $\sqrt{n-1}$ is what we scaled the singular values by to get the right units. The data in the new coordinates is $Y = U^TX$.

The spring-mass system

The solution to the displacement of mass in the spring-mass system is given by $z(t) = A\cos(\omega t + \varphi)$ where A and φ are determined by the initial displacement and velocity of the mass and the frequency ω can be found from the constants in the differential equation. We can set up three cameras around the system and collect video. In the frame of each video, there are two dimensions. For camera 1, we can denote them (xa, ya) at each instance in time. Similarly, we can do the same for camera 2 and camera 3.To make vectors of all the position at each time, we make the matrix $x = \begin{bmatrix} xa \\ ya \\ xb \\ yb \\ xc \\ yc \end{bmatrix}$. In spring-mass system, there should only be one nonzero singular value to represent 1D motion. Since no video recording in perfect, we expect a little bit of noise. There might also be some movement in the other directions if the mass isn't started at a perfectly vertical trajectory. Thus, we shouldn't expect only one nonzero singular value, but one is significantly larger than all the others, representing the dominant up-down motion of the mass. The large singular values help us tell which directions are most important. If our camera isn't shot at the right angle, the first principal component will tell us we need to rotate it. If singular values are really small, we can choose to ignore them as noise.

(Reference: Amath482 lecture notes from Professor Jason Bramburger)

Algorithm implementation and development:

First, I used load function to read the files and used size function to get the number of frames of each file. At each shift by making a loop, I converted them into gray scale by rgb2gray command, set a threshold at 230(unit-8 type ranges from 0 to 255), and located the points to extract the bright coordinates by find and mean commands. I did the same for each of the three videos from the same test. Then I found the one with the smallest frame size by the min function and cut off their size into the min. Using the min size, I put the xy coordinates from the three movies into a column matrix and subtracted the mean to get the position. I applied the singular value decomposition to the column matrix dived by the square root of n-1(scale the singular value to get the right unit)by the svd command to get the U,S,V matrix. I multiplied the position matrix by the transpose of U to find the projection. I also employed the diag function to find the singular values and plotted into graphs. Finally, I plotted the original displacement and displacement after Principal Component Analysis to compare. I did the same procedure for all of the four cases-ideal cases, noisy cases, horizontal displacement, horizontal displacement and rotation.

Computational results:

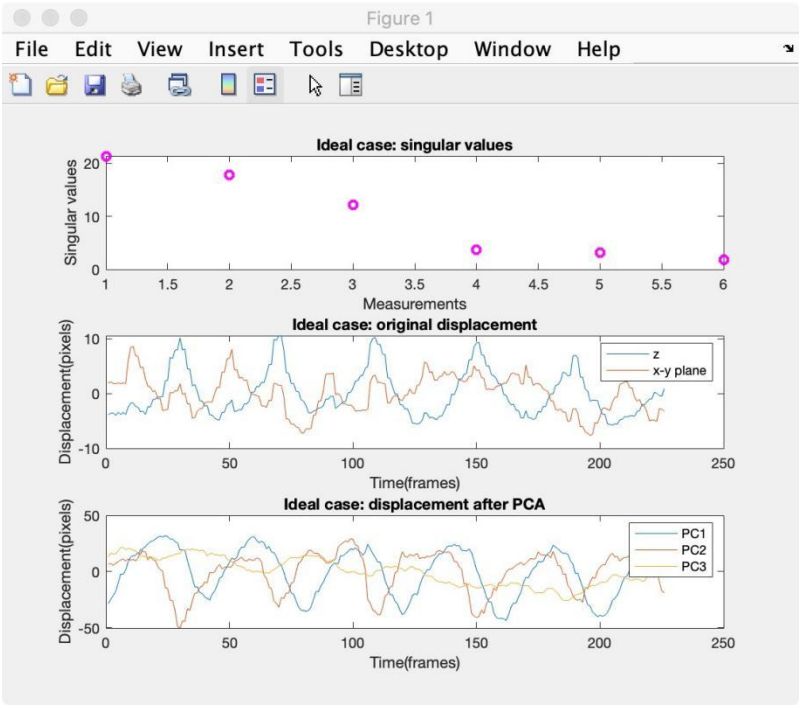

Figure 1: This figure displays the plot of the singular values(1), plot of the original displacement(2), and plot of the displacement after PCA(3) for the ideal case.

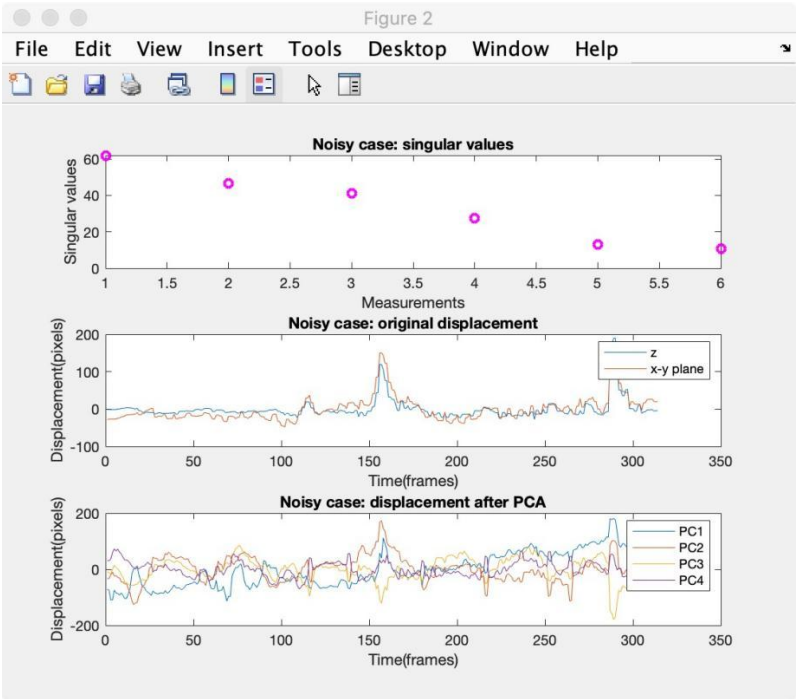

Figure 2: This figure displays the plot of the singular values(1), plot of the original displacement(2), and plot of the displacement after PCA(3) for the noisy case.

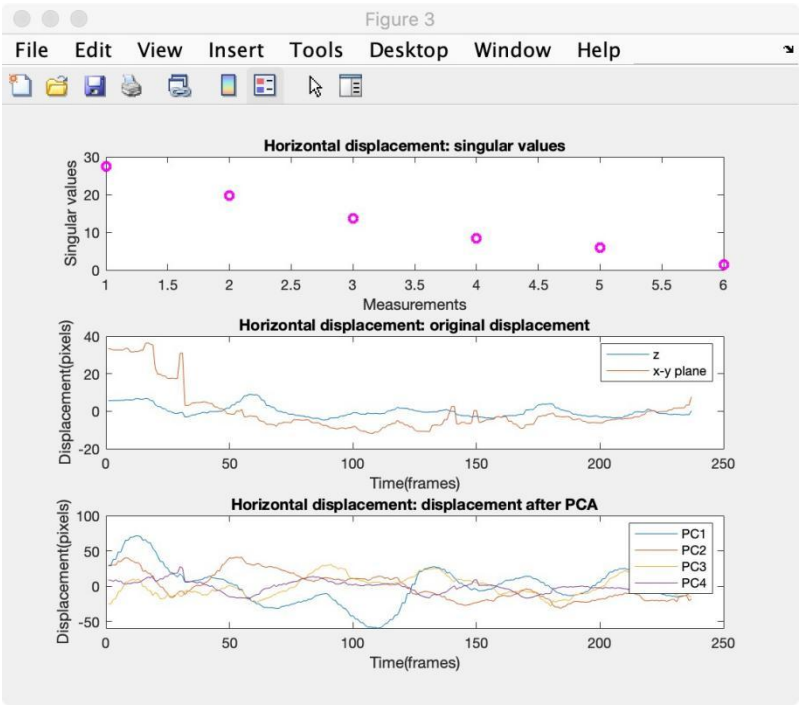

Figure 3: This figure displays the plot of the singular values(1), plot of the original displacement(2), and plot of the displacement after PCA(3) for the horizontal displacement.

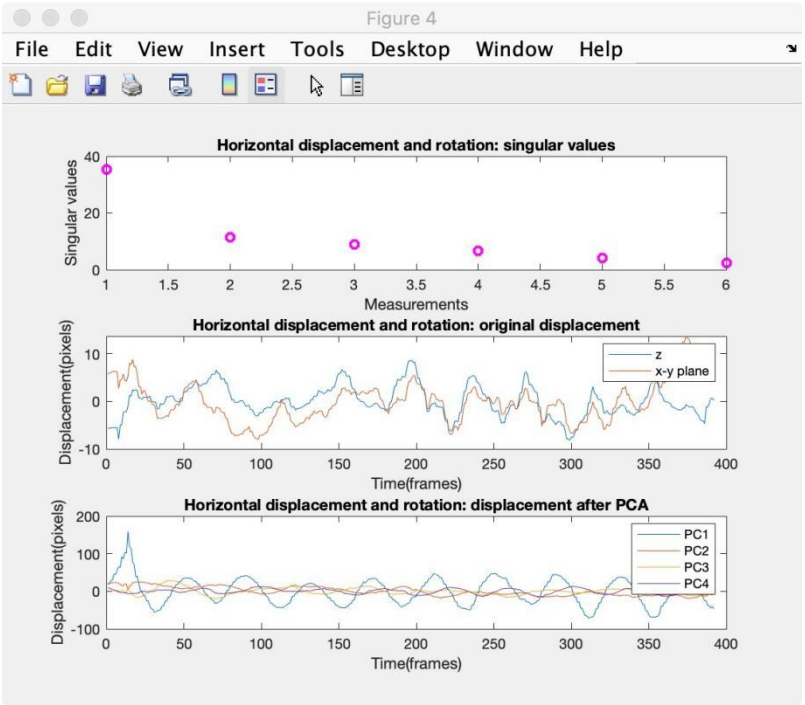

Figure 4: This figure displays the plot of the singular values(1), plot of the original displacement(2), and plot of the displacement after PCA(3) for the horizontal displacement and rotation.

In the ideal case, we can see from the figure that three principal components capture more than 90% of the energy. The projections are similar to the original data, especially the first two components. In the noisy case, with the shaking camera, we can see from the displacement that it oscillates badly.

We can also see from the figure that we have four principal components, which nearly capture 90% of the energy, meaning the noisy indeed impacts the singular value decomposition. In the horizontal displacement, we have four principal components captured 90%. It's worth noting that the displacement have fluctuation since the mass is released off-center. In the horizontal displacement and rotation, four principal components account for more than 90% energy. The projection onto the first principal component clearly show us the horizontal displacement and rotation acted on the mass at its first peak. From the figure, we can also assume that the PCA capture the nature of multidimensional horizontal displacement and rotation.

Summary and conclusions:

I applied the principal component analysis on the four different cases of the spring-mass system with the use of singular value decomposition. I made plots of the singular values, original displacement, and displacement after PCA of each test to compare and contrast the different cases.

MATLAB function used and brief implementation explanation:

-load: load variables from file into workspace
-size: return the length of the dimension
-find: return indices and values of nonzero elements
-rgb2gray: convert RGB image to grayscale
-min: return the minimum element of an array
-mean: return the mean of the elements
-[U,S,V] = svd(A,'econ'): perform a singular value decomposition and produce an economy-size decomposition
-diag: get diagonal elements of matrix
-plot: plot multiple XY pairs using the same axes
-subplot(m,n,p): divide the current figure into m*n grid and create axes in the position p
-set: set graphics object properties
-xlabel: label x-axis
-ylabel: label y-axis
-title: add title
-legend: add legend to axes
(Reference: MATLAB search helps)

MATLAB codes:

```
clear all; close all; clc


% Ideal case
load('cam1_1.mat')
numFrames1_1 = size(vidFrames1_1,4);
```

```
load('cam2_1.mat')
numFrames2_1 = size(vidFrames2_1,4);
load('cam3_1.mat')
numFrames3_1 = size(vidFrames3_1,4);

x1 = zeros(numFrames1_1);
y1 = zeros(numFrames1_1);
for j = 1:numFrames1_1
    X = vidFrames1_1(:,:,:,j);
    X = rgb2gray(X);
    [placex, placey] = find(X > 230);
    x1(j) = mean(placex);
    y1(j) = mean(placey);
end

x2 = zeros(numFrames2_1);
y2 = zeros(numFrames2_1);
for j = 1:numFrames2_1
    X = vidFrames2_1(:,:,:,j);
    X = rgb2gray(X);
    [placex, placey] = find(X > 230);
    x2(j) = mean(placex);
    y2(j) = mean(placey);
end

x3 = zeros(numFrames3_1);
y3 = zeros(numFrames3_1);
for j = 1:numFrames3_1
    X = vidFrames3_1(:,:,:,j);
    X = rgb2gray(X);
    [placex, placey] = find(X > 230);
    x3(j) = mean(placex);
    y3(j) = mean(placey);
end

min = min([numFrames1_1,numFrames2_1,numFrames3_1]);
xs = [x1(1:min);y1(1:min);x2(1:min);y2(1:min);x3(1:min);y3(1:min)];
position = xs - mean(xs,2);
[U,S,V] = svd(position/sqrt(min-1),'econ');
projection = U' * position;
sigma1 = diag(S);

figure(1)
subplot(3,1,1)
```

```
plot(1:6,sigma1,'mo','Linewidth',2);
xlabel('Measurements'); ylabel('Singular values');
title("Ideal case: singular values");
subplot(3,1,2)
plot(1:min, position(1,:), 1:min, position(2,:))
xlabel("Time(frames)"); ylabel("Displacement(pixels)");
title("Ideal case: original displacement");
legend("z", "x-y plane")
subplot(3,1,3)
plot(1:min,projection(1,:),1:min,projection(2,:),1:min,projection(3,:))
xlabel("Time(frames)"); ylabel("Displacement(pixels)");
title("Ideal case: displacement after PCA");
legend('PC1', 'PC2', 'PC3');

% Noisy case
load('cam1_2.mat')
numFrames1_2 = size(vidFrames1_2,4);
load('cam2_2.mat')
numFrames2_2 = size(vidFrames2_2,4);
load('cam3_2.mat')
numFrames3_2 = size(vidFrames3_2,4);

x1 = zeros(numFrames1_2);
y1 = zeros(numFrames1_2);
for j = 1:numFrames1_2
    X = vidFrames1_2(:,:,:,j);
    X = rgb2gray(X);
    [placex, placey] = find(X > 230);
    x1(j) = mean(placex);
    y1(j) = mean(placey);

end

x2 = zeros(numFrames2_2);
y2 = zeros(numFrames2_2);
for j = 1:numFrames2_2
    X = vidFrames2_2(:,:,:,j);
    X = rgb2gray(X);
    [placex, placey] = find(X > 230);
    x2(j) = mean(placex);
    y2(j) = mean(placey);
end

x3 = zeros(numFrames3_2);
```

```
y3 = zeros(numFrames3_2);
for j = 1:numFrames3_2
    X = vidFrames3_2(:,:,:,j);
    X = rgb2gray(X);
    [placex, placey] = find(X > 230);
    x3(j) = mean(placex);
    y3(j) = mean(placey);
end

min = min([numFrames1_2,numFrames2_2,numFrames3_2]);
xs = [x1(1:min);y1(1:min);x2(1:min);y2(1:min);x3(1:min);y3(1:min)];
position = xs - mean(xs,2);
[U,S,V] = svd(position/sqrt(min-1),'econ');
projection = U' * position;
sigma2 = diag(S);

figure(2)
subplot(3,1,1)
plot(1:6,sigma2,'mo','Linewidth',2);
xlabel('Measurements'); ylabel('Singular values');
title("Noisy case: singular values");
subplot(3,1,2)
plot(1:min, position(1,:), 1:min, position(2,:))
xlabel("Time(frames)"); ylabel("Displacement(pixels)");
title("Noisy case: original displacement");
legend("z", "x-y plane")
subplot(3,1,3)
plot(1:min,projection(1,:),1:min,projection(2,:),1:min,projection(3,:),1:min,projection(4,:))
xlabel("Time(frames)"); ylabel("Displacement(pixels)");
title("Noisy case: displacement after PCA");
legend('PC1', 'PC2', 'PC3', 'PC4');

% Horizontal displacement
load('cam1_3.mat')
numFrames1_3 = size(vidFrames1_3,4);
load('cam2_3.mat')
numFrames2_3 = size(vidFrames2_3,4);
load('cam3_3.mat')
numFrames3_3 = size(vidFrames3_3,4);

x1 = zeros(numFrames1_3);
y1 = zeros(numFrames1_3);
for j = 1:numFrames1_3
    X = vidFrames1_3(:,:,:,j);
```

```
    X = rgb2gray(X);
    [placex, placey] = find(X > 230);
    x1(j) = mean(placex);
    y1(j) = mean(placey);

end

x2 = zeros(numFrames2_3);
y2 = zeros(numFrames2_3);
for j = 1:numFrames2_3
    X = vidFrames2_3(:,:,:,j);
    X = rgb2gray(X);
    [placex, placey] = find(X > 230);
    x2(j) = mean(placex);
    y2(j) = mean(placey);
end

x3 = zeros(numFrames3_3);
y3 = zeros(numFrames3_3);
for j = 1:numFrames3_3
    X = vidFrames3_3(:,:,:,j);
    X = rgb2gray(X);
    [placex, placey] = find(X > 230);
    x3(j) = mean(placex);
    y3(j) = mean(placey);
end

min = min([numFrames1_3,numFrames2_3,numFrames3_3]);
xs = [x1(1:min);y1(1:min);x2(1:min);y2(1:min);x3(1:min);y3(1:min)];
position = xs - mean(xs,2);
[U,S,V] = svd(position/sqrt(min-1),'econ');
projection = U' * position;
sigma3 = diag(S);

figure(3)
subplot(3,1,1)
plot(1:6,sigma3,'mo','Linewidth',2);
xlabel('Measurements'); ylabel('Singular values');
title("Horizontal displacement: singular values");
subplot(3,1,2)
plot(1:min, position(1,:), 1:min, position(2,:))
xlabel("Time(frames)"); ylabel("Displacement(pixels)");
title("Horizontal displacement: original displacement");
legend("z", "x-y plane")
```

```
subplot(3,1,3)
plot(1:min,projection(1,:),1:min,projection(2,:),1:min,projection(3,:),1:min,projection(4,:))
xlabel("Time(frames)"); ylabel("Displacement(pixels)");
title("Horizontal displacement: displacement after PCA");
legend('PC1', 'PC2', 'PC3', 'PC4');

% Horizontal displacement and rotation
load('cam1_4.mat')
numFrames1_4 = size(vidFrames1_4,4);
load('cam2_4.mat')
numFrames2_4 = size(vidFrames2_4,4);
load('cam3_4.mat')
numFrames3_4 = size(vidFrames3_4,4);

x1 = zeros(numFrames1_4);
y1 = zeros(numFrames1_4);
for j = 1:numFrames1_4
    X = vidFrames1_4(:,:,:,j);
    X = rgb2gray(X);
    [placex, placey] = find(X > 230);
    x1(j) = mean(placex);
    y1(j) = mean(placey);

end

x2 = zeros(numFrames2_4);
y2 = zeros(numFrames2_4);
for j = 1:numFrames2_4
    X = vidFrames2_4(:,:,:,j);
    X = rgb2gray(X);
    [placex, placey] = find(X > 230);
    x2(j) = mean(placex);
    y2(j) = mean(placey);
end

x3 = zeros(numFrames3_4);
y3 = zeros(numFrames3_4);
for j = 1:numFrames3_4
    X = vidFrames3_4(:,:,:,j);
    X = rgb2gray(X);
    [placex, placey] = find(X > 230);
    x3(j) = mean(placex);
    y3(j) = mean(placey);
end
```

```
min = min([numFrames1_4,numFrames2_4,numFrames3_4]);
xs = [x1(1:min);y1(1:min);x2(1:min);y2(1:min);x3(1:min);y3(1:min)];
position = xs - mean(xs,2);
[U,S,V] = svd(position/sqrt(min-1),'econ');
projection = U' * position;
sigma4 = diag(S);

figure(4)
subplot(3,1,1)
plot(1:6,sigma4,'mo','Linewidth',2);
xlabel('Measurements'); ylabel('Singular values');
title("Horizontal displacement and rotation: singular values");
subplot(3,1,2)
plot(1:min, position(1,:), 1:min, position(2,:))
xlabel("Time(frames)"); ylabel("Displacement(pixels)");
title("Horizontal displacement and rotation: original displacement");
legend("z", "x-y plane")
subplot(3,1,3)
plot(1:min,projection(1,:),1:min,projection(2,:),1:min,projection(3,:),1:min,projection(4,:))
xlabel("Time(frames)"); ylabel("Displacement(pixels)");
title("Horizontal displacement and rotation: displacement after PCA");
legend('PC1', 'PC2', 'PC3', 'PC4');
```