Principal Component Analysis

Author: Seoyoung Park

Abstract

A Singular Value Decomposition is significant tool in broad areas of applications because of its a lot of mathematical properties and the fact that it exists in every matrix. The SVD method allows every matrix to be diagonalized with the proper bases for the domain and range. Principal Component Analysis is one of applications of Singular Value Decomposition that provides a least-square fitting algorithm from complex dataset and lets us to project the matrix into low-dimensional representations.

Introduction and Overview

In this paper, the goal is to illustrate various aspects of the PCA and its practical usefulness and the effects of noise on the PCA algorithms with given movie files created from three different cameras (videos are from 2011). Three different cameras are used to extract out data concerning the behavior of the paint can and get the governing equations of motion. There are 4 cases tested with the paint can. First one is an ideal case that the entire motion is in the z direction with simple harmonic motion (camN1.mat where N = 1,2,3). Next, a noisy case is tested that the ideal case is repeated but with shake of camera into the video recording (camN2.mat where N = 1,2,3). Also, in the horizontal displacement case, the paint can is released off-center so as to produce motion in the x,y plane as well as the z direction, so both a pendulum motion and a simple harmonic oscillations exist in this case (camN3.mat where N=1,2,3). Lastly, in the horizontal displacement and rotation case, the previous case is repeated with rotations in x-y plane and z direction (camN4.mat where N=1,2,3).

Theoretical Background

A singular value decomposition (SVD) is a matrix factorization that provides a transformation of the given set of vectors to stretch or compress and rotate. If the matrix A is an $m \times n$ matrix (m < n), $Av_j = \sigma_j u_j$ $(1 \leq j \leq n)$. The form of SVD decomposition is

$$AV = U\Sigma$$

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

(U and V are unitary, and Σ is diagonal)

The matrix A applies a unitary transformation(U^*), then Σ stretches and creates an ellipse with principal semi axes, and U rotates the created hyper-ellipse. Now, if we consider there is a simple mass-spring system so that we are trying to record the motion of the mass with three cameras. Assume the mass is moving by a small perturbation in the z-direction. Since we have 3 cameras to observe the motion, the data sets would have redundancy and noise. However, we are able to get a simple version of the data sets with principal component analysis (PCA). Let's say each camera gives a two-dimensional data such as camera 1 : (x_a, y_a), camera 2: (x_b, y_b), camera 3: (x_c, y_c). These can be expressed in a data matrix

$$X = \begin{bmatrix} x_a \\ y_a \\ x_b \\ y_b \\ x_c \\ y_c \end{bmatrix} \in R^{m \times n}$$

(m: the number of measurement types, n: the number of data points the camera take over time)

Since we got the data set from several perspectives, we can get an idea of redundancy with a covariance. If the value of covariance is very small, it means the pair is statistically independent; otherwise, it means statistically dependent, which gives redundancy. A covariance matrix can be expressed by

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

C_X is 6×6 matrix with diagonal entries of variance measures and off-diagonal entries of covariance between all pairs. The main purpose here is to remove redundancy by making all off-diagonal entries to be zero and see what variance has maximum value. This can be identified by diagonalizing the matrix C, and this is why SVD matters in PCA since ,as mentioned above, SVD provides diagonal matrix from any matrix. Note that SVD has two bases, U and V. For PCA, we can set principal component basis $Y = U^* X$, then the covariance matrix of Y is following

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

In a new basis matrix Y, the principal components are the columns of U and variances are the elements of Σ^2

Algorithm Implementation and Development

With given video datasets, we are going to compare four cases and see what PCA tells us about the system.

Tracking Object from Videos

1. Load the video
   Since the videos are in .mat format, we can just load the videos and find the size of each video which tells us how many frames are in each video. Since there are 3 different cameras, set the minimum number of frames of video of 3 cameras to be the fixed number of frames for all 3 cameras, so it would be easier to compare. Also, reduce the dimension of the matrix from 4D.

2. Convert each frame to gray-scale
   Converting the given matrix into gray-scale will result in 3D matrix which is 2D matrix of each frame and time by running the loop backwards from the number of frames down to 1 to ensure that frame matrix is initialized to its final size the first time through the loop.

3. Crop the region of interest
   To reduce background noise and to increase the accuracy of the result, select the row and column range by watching videos.

4. Compute frame differences Compute frame differences using imabsdiff. Compute a threshold that divides an image into background and foreground pixels and compute the absolute difference between each frame and its corresponding background estimate.

5. Compute the location of the can in each frame
   Label each individual object (using logical) and compute its corresponding center of mass (using regionprops) then compute the location of the can in each frame, assuming that the paint can is the largest object in each frame.

Filtering and PCA

1. Apply Gaussian filter
   Smooth a vector of noisy data with a Gaussian-weighted moving average filter, so it looks more simple harmonic motion.

2. Construct a data matrix X
   Construct the data matrix X with six rows that correspond to each x and y direction of the motion of the paint can captured by 3 different cameras, then subtract the mean of each row to normalize the matrix.

3. Perform SVD
   To get the principal components, perform SVD of data matrix X divided by $\sqrt{n-1}$ where n is the number of columns of X. Once we get two bases and one diagonal matrix, obtain diagonal values of Σ with diag and square it. We can get scale of principal component by divide the diagonal values by sum of all diagonal values.

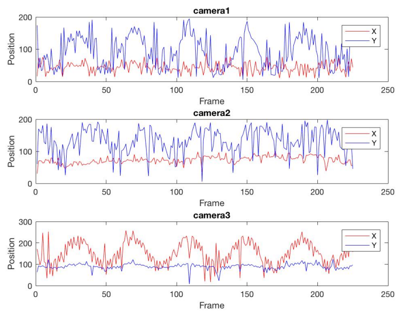
4. Get the principal component basis matrix Y
   Now we can get the principal component basis Y, the projection of original data, by multiplying transpose of U matrix with data matrix. Y, a new basis, shows the new displacement of the paint can obtained by three different cameras.

Computational Results

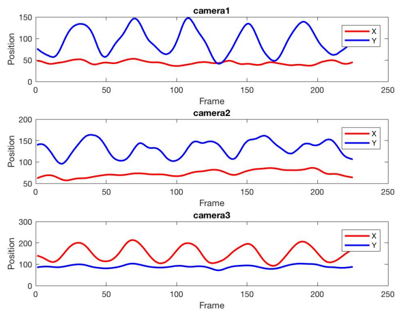
After following the procedure shown above for each 4 cases, we can compare how those 4 cases look different on the graphs.

Case 1: Ideal Case

In this case, the paint can has a simple harmonic motion in the z direction and the ensuing oscillations. Since the video taken by the camera 3 is rotated 90 degree, the overall displacements of x and y captured by camera 3 are different from those captured by the other cameras. After filtering with gaussian, the data looks more clear and more like simple harmonic motion. PCA algorithm results in three major modes, and other modes are close to zero as shown in figure 2. Since the diagonal matrix that has singular values are in order from greatest to least, the very first value is the greatest which tells us a quantification of the importance of the mode. We can reconstruct the data with 1 mode, which is the most important, and 3 major modes. Since we already got the unitary matrix U, we can get the new projection of an original data Y which illustrates new x, y displacement of the paint can.

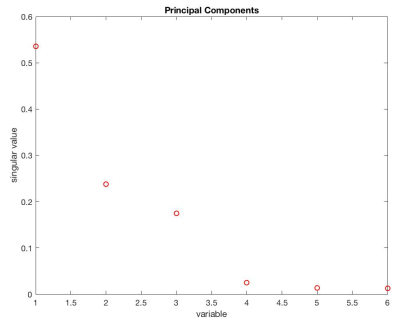


(a) original

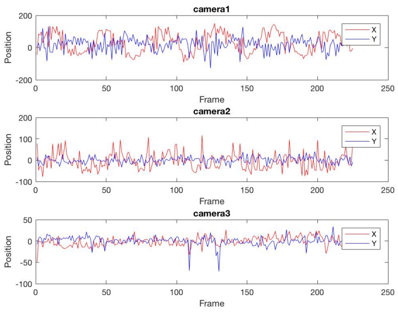


(b) gaussian filter

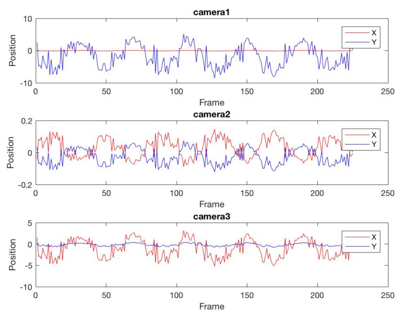
Figure 1: Shows the displacement of the paint can of x and y direction.



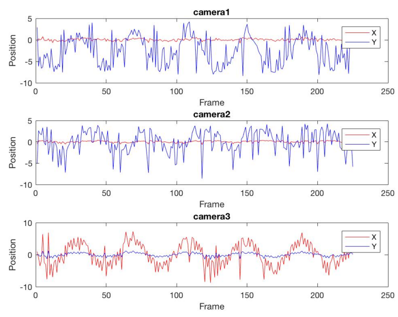
(a) Principal components corresponding to singular values.



(b) The new basis Y that shows the new displacement of the paint can



(c) 1 mode

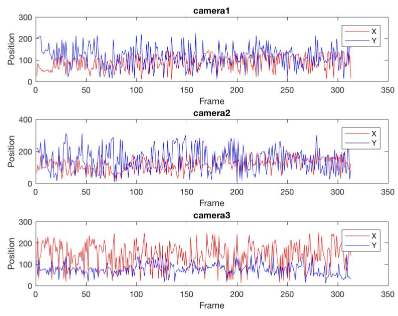


(d) 3 mode

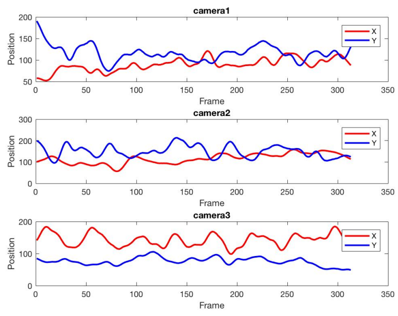
Figure 2: 1 mode and 3 mode approximations; 3 mode approximation looks more similar to the original dataset, but still 1 mode approximation looks similar to original as well

Case 2: Noisy Case

In this case, repeat the ideal case experiment, but introduce camera shake into the video recording as clearly shown in figure 5 compared to figure 1. Due to a lot of noise, it is hard to capture simple harmonic motion even after applying gaussian filter, more principal components are significant then the ideal case is (compare figure 3 and figure 6(a)), and new basis looks more messy as well.

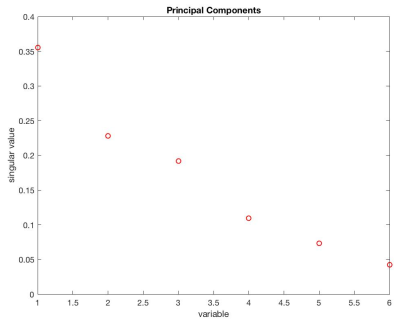


(a) original

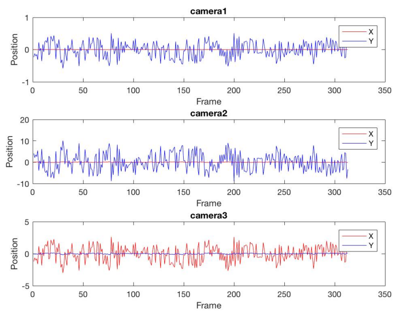


(b) gaussian filter

Figure 3: Shows the displacement of the paint can of x and y direction.



(a) Principal Components corresponding to singular values



(b) Reconstruct with 1 mode which is the most significant

Figure 4

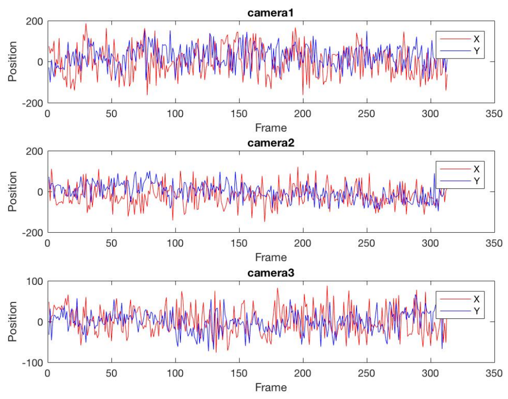
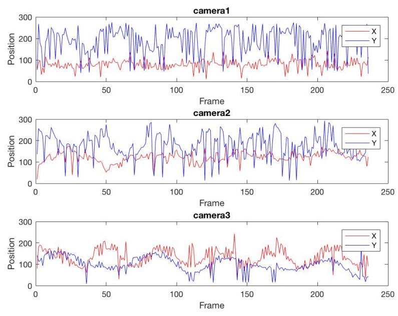


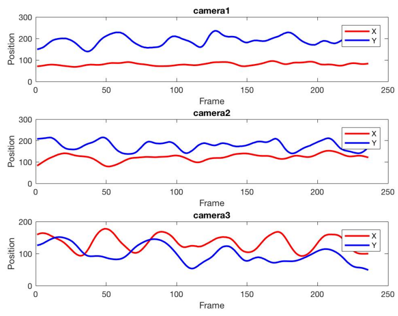
Figure 5: The new basis Y that shows the new displacement of the paint can

Case 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. Thus there is both a pendulum motion and a simple harmonic oscillations. As well as in this case, more than one principal components are required, but still there are some modes that have relatively high singular value than others (as shown in figure 9(a)), PCA picks up some noise.

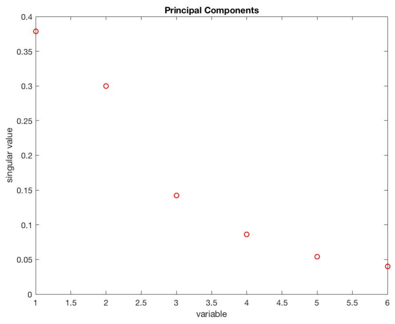


(a) original

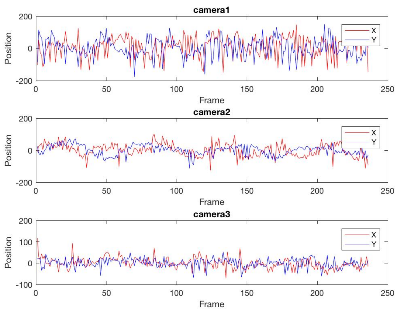


(b) gaussian filter

Figure 6: Shows the displacement of the paint can of x and y direction.



(a) Principal Components corresponding to singular values

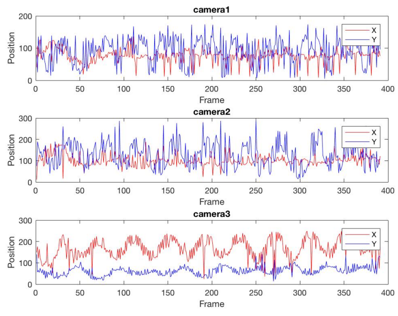


(b) The new basis Y that shows the new displacement of the paint can

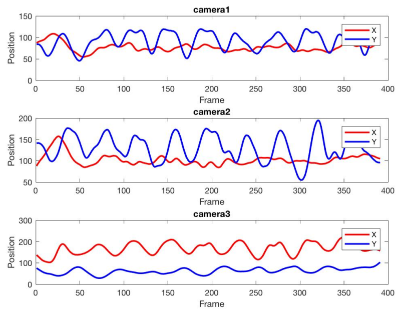
Figure 7

Case 4: Horizontal Displacement and Rotation

In this case, the mass is released off-center and rotates so as to produce motion in the x−y plane, rotation as well as in the z direction. Thus there is both a pendulum motion and a simple harmonic oscillations. If you compare principal components from figure 9(a) and 11(a) there is no much difference because PCA cannot capture the rotation since the rotation is nonlinear.

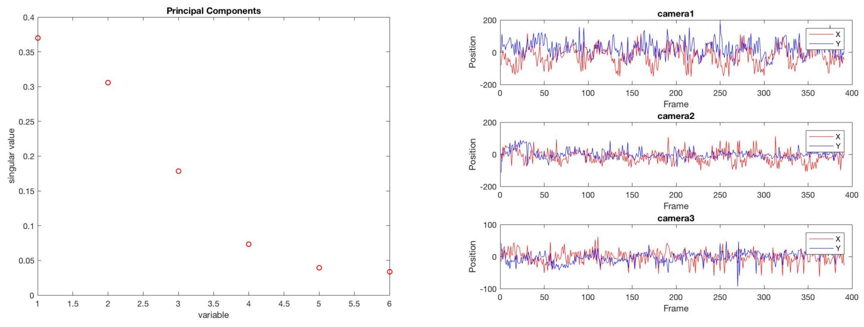


(a) original



(b) gaussian filter

Figure 8: Shows the displacement of the paint can of x and y direction.



(a) Principal Components corresponding to singular values

(b) The new basis of dataset that shows the displacement of the paint can

Figure 9

Summary and Conclusions

Understanding the singular value decomposition, we can apply it to principal component to analyze a number of videos of a simple mass spring system taken by 3 different cameras with 4 different cases. After applying PCA to all cases, PCA allows us to remove redundancy and get new basis, projection of the original data.

Reference

- "A." Smooth noisy data - MATLAB smoothdata, www.mathworks.com/help/matlab/ref/smoothdata.html.
- Alex Dytso Alex Dytso (view profile) 22 files 58 downloads 2.72917. "Select Your Country." Extended Kalman Filter Tracking Object in 3-D - File Exchange - MATLAB Central, 23 Apr. 2012, www.mathworks.com/matlabcentral/fileexchange/36301-extended-kalman-filter-tracking-object-in-3-d?focused=5227951&tab=function.

Appendix A

rgb2gray: convert RGB image or colormap to grayscale

Z = imabsdiff(X,Y): subtracts each element in array Y from the corresponding element in array X and returns the absolute difference in the corresponding element of the output array Z

BW = imbinarize(I): binarizes image I with a global threshold computed using Otsu's method, which chooses the threshold to minimize the intraclass variance of the thresholded black and white pixels

LEVEL = graythresh(I): computes a global threshold (LEVEL) that can be used to convert an intensity image to a binary image with IMBINARIZE

STATS = regionprops(BW,PROPERTIES): measures a set of properties for each connected component (object) in the binary image BW, which must be a logical array; it can have any dimension

centroid: finds the centroid of a polyshape

B = smoothdata(A): for a vector A returns a smoothed version of A using a moving average with a fixed window length.

[U,S,V] = svd(X) produces a diagonal matrix S, of the same dimension as X and with nonnegative diagonal elements in decreasing order, and unitary matrices U and V so that X = U*S*V'.

Appendix B

```
clear all; close all; clc;

%%Ideal Case

% camera 1

load cam1_1.mat % load the video
num1 = size(vidFrames1_1,4);
load cam2_1.mat
num2 = size(vidFrames2_1,4);
load cam3_1.mat
num3 = size(vidFrames3_1,4);
numframes = min([num1 num2 num3]);

% for i = 1:num1
%     frame1 = rgb2gray(vidFrames1_1(:,:,:,i)); % convert color video into gray scale
%     %imshow(frame1(200:400,300:400)); % see if the cropped one looks good
% end

% Running the loop backwards from numframes down to 1, is a common MATLAB programming trick
% to ensure that g is initialized to its final size the first time through the loop
for k = numframes:-1:1
frame1(:, :, k) = rgb2gray(vidFrames1_1(:, :, :, k)); % convert each frame to grayscale using rgb2
end
g1 = frame1(200:400,300:400,:);

for k = numframes-1:-1:1
d1(:, :, k) = imabsdiff(g1(:, :, k), g1(:, :, k+1)); % compute frame differences using imabsdiff
end
%imtool(d1(:, :, 1), [])

% computes a threshold that divides an image into background and foreground pixels
% compute the absolute difference between each frame and its corresponding background estimate
for k = numframes-1:-1:1
thresh1(:,:,k) = imbinarize(d1(:,:,k),graythresh(d1(:,:,k)));
end

% label each individual object (using logical) and compute its corresponding center of mass (using
% compute the location of the can in each frame
% assuming that the can is the largest object in each frame
centroids1 = zeros(numframes, 2);
for k = numframes-1:-1:1
s1 = regionprops(logical(thresh1(:, :, k)), 'area', 'centroid');
area_vector1 = [s1.Area];
[tmp1, idx1] = max(area_vector1);
centroids1(k, :) = s1(idx1(1)).Centroid;
x1(k) = centroids1(k,1);
y1(k) = centroids1(k,2);
end
```

```
xavg1 = mean(x1,2);
yavg1 = mean(y1,2);

%Smooth a vector of noisy data with a Gaussian-weighted moving average filter
smoothx1 = smoothdata(x1,'gaussian',20);
smoothy1 = smoothdata(y1,'gaussian',20);

% camera 2

% for i = 1:num2
%     frame2 = rgb2gray(vidFrames2_1(:,:,:,i)); % convert color video into gray scale
%     %imshow(frame2(100:300,240:350))
% end

for k = numframes:-1:1
frame2(:, :, k) = rgb2gray(vidFrames2_1(:, :, :, k)); % convert each frame to grayscale using rgb2
end
g2 = frame2(100:300, 240:350, :);
for k = numframes-1:-1:1
d2(:, :, k) = imabsdiff(g2(:, :, k), g2(:, :, k+1)); % compute frame differences using imabsdiff
end
%imtool(d1(:, :, 1), [])

% computes a threshold that divides an image into background and foreground pixels
% compute the absolute difference between each frame and its corresponding background estimate
for k = numframes-1:-1:1
thresh2(:,:,k) = imbinarize(d2(:,:,k),graythresh(d2(:,:,k)));
end

% label each individual object (using logical) and compute its corresponding center of mass (using
% compute the location of the can in each frame
% assuming that the can is the largest object in each frame
centroids2 = zeros(numframes, 2);
for k = numframes-1:-1:1
s2 = regionprops(logical(thresh2(:, :, k)), 'area', 'centroid');
area_vector2 = [s2.Area];
[tmp2, idx2] = max(area_vector2);
centroids2(k, :) = s2(idx2(1)).Centroid;
x2(k) = centroids2(k,1);
y2(k) = centroids2(k,2);
end
xavg2 = mean(x2,2);
yavg2 = mean(y2,2);

%Smooth a vector of noisy data with a Gaussian-weighted moving average filter
smoothx2 = smoothdata(x2,'gaussian',20);
smoothy2 = smoothdata(y2,'gaussian',20);

% camera 3

% for i = 1:num3
%     frame3 = rgb2gray(vidFrames3_1(:,:,:,i)); % convert color video into gray scale
%     %imshow(frame3(200:350,240:500))
% end

for k = numframes:-1:1
frame3(:, :, k) = rgb2gray(vidFrames3_1(:, :, :, k)); % convert each frame to grayscale using rgb2
end
```

```
g3 = frame3(200:350, 240:500, :);
for k = numframes-1:-1:1
d3(:, :, k) = imabsdiff(g3(:, :, k), g3(:, :, k+1)); % compute frame differences using imabsdiff
end
%imtool(d1(:, :, 1), [])

% computes a threshold that divides an image into background and foreground pixels
% compute the absolute difference between each frame and its corresponding background estimate
for k = numframes-1:-1:1
thresh3(:,:,k) = imbinarize(d3(:,:,k),graythresh(d3(:,:,k)));
end

% label each individual object (using logical) and compute its corresponding center of mass (using
% compute the location of the can in each frame
% assuming that the can is the largest object in each frame
centroids3 = zeros(numframes, 2);
for k = numframes-1:-1:1
s3 = regionprops(logical(thresh3(:, :, k)), 'area', 'centroid');
area_vector3 = [s3.Area];
[tmp3, idx3] = max(area_vector3);
centroids3(k, :) = s3(idx3(1)).Centroid;
x3(k) = centroids3(k,1);
y3(k) = centroids3(k,2);
end
xavg3 = mean(x3,2);
yavg3 = mean(y3,2);

%Smooth a vector of noisy data with a Gaussian-weighted moving average filter
smoothx3 = smoothdata(x3,'gaussian',20);
smoothy3 = smoothdata(y3,'gaussian',20);

frame = 1:225;

% plot the xy position
figure(1);
subplot(3,1,1), plot(frame, x1,'r', 1:225, y1,'b')
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, x2,'r', 1:225, y2,'b')
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, x3,'r', 1:225, y3,'b')
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');

% plot the smooth-xy position (more look like simple harmonic motion)
figure(2);
subplot(3,1,1), plot(frame, smoothx1,'r', frame, smoothy1,'b', 'Linewidth',[2])
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, smoothx2,'r', 1:225, smoothy2,'b', 'Linewidth',[2])
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, smoothx3,'r', 1:225, smoothy3,'b', 'Linewidth',[2])
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');


X = [x1-repmat(xavg1,1,225);
    y1-repmat(yavg2,1,225);
    x2-repmat(xavg2,1,225);
    y2-repmat(yavg2,1,225);
    x3-repmat(xavg3,1,225);
    y3-repmat(yavg3,1,225)]; % data matrix (subtracting mean)
```

```
[m n] = size(X);

[u s v] = svd(X/sqrt(n-1)); % svd

lambda = diag(s).^2; % diagonal matrix with eigenvalues of XX^T

Y = u' * X; % principal component basis

for j = 1:3
    X2 = u(:,1:j)*s(1:j,1:j)*v(:,1:j)';
end

X3 = u(:,1)*s(1,1)*v(:,1)';

% plot principal component
figure(3);
plot(lambda/sum(lambda),'ro'), title('Principal Components'); xlabel('variable'); ylabel('singular



% plot 3 modes
figure(4);
subplot(3,1,1), plot(frame, X2(1,:),'r', frame, X2(2,:),'b')
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, X2(3,:),'r', frame, X2(4,:),'b')
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, X2(5,:),'r', frame, X2(6,:),'b')
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');

% Y
figure(5);
subplot(3,1,1), plot(frame, Y(1,:),'r', frame, Y(2,:),'b')
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, Y(3,:),'r', frame, Y(4,:),'b')
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, Y(5,:),'r', frame, Y(6,:),'b')
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');

% plot 1 mode
figure(6);
subplot(3,1,1), plot(frame, X3(1,:),'r', frame, X3(2,:),'b')
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, X3(3,:),'r', frame, X3(4,:),'b')
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, X3(5,:),'r', frame, X3(6,:),'b')
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');


clear all; close all; clc;

%%Noisy Case

% camera 1

load cam1_2.mat % load the video
num1 = size(vidFrames1_2,4);
load cam2_2.mat
num2 = size(vidFrames2_2,4);
```

```
load cam3_2.mat
num3 = size(vidFrames3_2,4);
numframes = min([num1 num2 num3]);

% for i = 1:num1
%     frame1 = rgb2gray(vidFrames1_2(:,:,:,i)); % convert color video into gray scale
%     imshow(frame1(200:430,300:450)) % see if the cropped one looks good
% end

for k = numframes:-1:1
frame1(:, :, k) = rgb2gray(vidFrames1_2(:, :, :, k)); % convert each frame to grayscale using rgb2
end
g1 = frame1(200:430,300:450,:);

for k = numframes-1:-1:1
d1(:, :, k) = imabsdiff(g1(:, :, k), g1(:, :, k+1)); % compute frame differences using imabsdiff
end

for k = numframes-1:-1:1
thresh1(:,:,k) = imbinarize(d1(:,:,k),graythresh(d1(:,:,k)));
end

centroids1 = zeros(numframes, 2);
for k = numframes-1:-1:1
s1 = regionprops(logical(thresh1(:, :, k)), 'area', 'centroid');
area_vector1 = [s1.Area];
[tmp1, idx1] = max(area_vector1);
centroids1(k, :) = s1(idx1(1)).Centroid;
x1(k) = centroids1(k,1);
y1(k) = centroids1(k,2);
end

xavg1 = mean(x1,2);
yavg1 = mean(y1,2);
smoothx1 = smoothdata(x1,'gaussian',20);
smoothy1 = smoothdata(y1,'gaussian',20);

% camera 2
%
% for i = 1:num2
%     frame2 = rgb2gray(vidFrames2_2(:,:,:,i)); % convert color video into gray scale
%     imshow(frame2(70:400,200:400))
% end

for k = numframes:-1:1
frame2(:, :, k) = rgb2gray(vidFrames2_2(:, :, :, k)); % convert each frame to grayscale using rgb2
end
g2 = frame2(70:400, 200:400, :);

for k = numframes-1:-1:1
d2(:, :, k) = imabsdiff(g2(:, :, k), g2(:, :, k+1)); % compute frame differences using imabsdiff
end

for k = numframes-1:-1:1
thresh2(:,:,k) = imbinarize(d2(:,:,k),graythresh(d2(:,:,k)));
end

centroids2 = zeros(numframes, 2);
```

```
for k = numframes-1:-1:1
s2 = regionprops(logical(thresh2(:, :, k)), 'area', 'centroid');
area_vector2 = [s2.Area];
[tmp2, idx2] = max(area_vector2);
centroids2(k, :) = s2(idx2(1)).Centroid;
x2(k) = centroids2(k,1);
y2(k) = centroids2(k,2);
end

xavg2 = mean(x2,2);
yavg2 = mean(y2,2);
smoothx2 = smoothdata(x2,'gaussian',20);
smoothy2 = smoothdata(y2,'gaussian',20);

% camera 3
%
% for i = 1:num3
%     frame3 = rgb2gray(vidFrames3_2(:,:,:,i)); % convert color video into gray scale
%     imshow(frame3(200:350,240:500))
% end

for k = numframes:-1:1
frame3(:, :, k) = rgb2gray(vidFrames3_2(:, :, :, k)); % convert each frame to grayscale using rgb2
end
g3 = frame3(200:350, 240:500, :);

for k = numframes-1:-1:1
d3(:, :, k) = imabsdiff(g3(:, :, k), g3(:, :, k+1)); % compute frame differences using imabsdiff
end

for k = numframes-1:-1:1
thresh3(:,:,k) = imbinarize(d3(:,:,k),graythresh(d3(:,:,k)));
end

centroids3 = zeros(numframes, 2);
for k = numframes-1:-1:1
s3 = regionprops(logical(thresh3(:, :, k)), 'area', 'centroid');
area_vector3 = [s3.Area];
[tmp3, idx3] = max(area_vector3);
centroids3(k, :) = s3(idx3(1)).Centroid;
x3(k) = centroids3(k,1);
y3(k) = centroids3(k,2);
end

xavg3 = mean(x3,2);
yavg3 = mean(y3,2);
smoothx3 = smoothdata(x3,'gaussian',20);
smoothy3 = smoothdata(y3,'gaussian',20);

frame = 1:313;

% plot the xy position
figure(1);
subplot(3,1,1), plot(frame, x1,'r', frame, y1,'b')
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, x2,'r', frame, y2,'b')
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, x3,'r', frame, y3,'b')
```

```
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');

% plot the smooth-xy position
figure(2);
subplot(3,1,1), plot(frame, smoothx1,'r', frame, smoothy1,'b', 'Linewidth',[2])
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, smoothx2,'r', frame, smoothy2,'b', 'Linewidth',[2])
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, smoothx3,'r', frame, smoothy3,'b', 'Linewidth',[2])
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');

X = [x1-repmat(xavg1,1,313);
    y1-repmat(yavg2,1,313);
    x2-repmat(xavg2,1,313);
    y2-repmat(yavg2,1,313);
    x3-repmat(xavg3,1,313);
    y3-repmat(yavg3,1,313)]; % data matrix
[m n] = size(X);

[u s v] = svd(X/sqrt(n-1),'econ'); % svd
lambda = diag(s).^2; % diagonal matrix with eigenvalues of XX^T
Y = u' * X; % principal component basis

X2 = u(:,1)*s(1,1)*v(:,1)';

% plot principal component
figure(3);
plot(lambda/sum(lambda),'ro'), title('Principal Components'); xlabel('variable'); ylabel('singular

% plot Y
figure(4);
subplot(3,1,1), plot(frame, Y(1,:),'r', frame, Y(2,:),'b')
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, Y(3,:),'r', frame, Y(4,:),'b')
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, Y(5,:),'r', frame, Y(6,:),'b')
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');

% plot 1mode
figure(5);
subplot(3,1,1), plot(frame, X2(1,:),'r', frame, X2(2,:),'b')
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, X2(3,:),'r', frame, X2(4,:),'b')
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, X2(5,:),'r', frame, X2(6,:),'b')
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');


clear all; close all; clc;

%%horizontal displacement

load cam1_3.mat % load the video
num1 = size(vidFrames1_3,4);
load cam2_3.mat
num2 = size(vidFrames2_3,4);
load cam3_3.mat
num3 = size(vidFrames3_3,4);
numframes = min([num1 num2 num3]);
```

```
% camera 1

% for i = 1:num1
%     frame1 = rgb2gray(vidFrames1_3(:,:,:,i)); % convert color video into gray scale
%     imshow(frame1(120:400,250:400)) % see if the cropped one looks good
% end

for k = numframes:-1:1
frame1(:, :, k) = rgb2gray(vidFrames1_3(:, :, :, k)); % convert each frame to grayscale using rgb2
end

g1 = frame1(120:400,250:400,:);
for k = numframes-1:-1:1
d1(:, :, k) = imabsdiff(g1(:, :, k), g1(:, :, k+1)); % compute frame differences using imabsdiff
end

for k = numframes-1:-1:1
thresh1(:,:,k) = imbinarize(d1(:,:,k),graythresh(d1(:,:,k)));
end

centroids1 = zeros(numframes, 2);
for k = numframes-1:-1:1
s1 = regionprops(logical(thresh1(:, :, k)), 'area', 'centroid');
area_vector1 = [s1.Area];
[tmp1, idx1] = max(area_vector1);
centroids1(k, :) = s1(idx1(1)).Centroid;
x1(k) = centroids1(k,1);
y1(k) = centroids1(k,2);
end

xavg1 = mean(x1,2);
yavg1 = mean(y1,2);
smoothx1 = smoothdata(x1,'gaussian',20);
smoothy1 = smoothdata(y1,'gaussian',20);

% camera 2

% for i = 1:num2
%     frame2 = rgb2gray(vidFrames2_3(:,:,:,i)); % convert color video into gray scale
%     imshow(frame2(100:400,200:400))
% end

for k = numframes:-1:1
frame2(:, :, k) = rgb2gray(vidFrames2_3(:, :, :, k)); % convert each frame to grayscale using rgb2
end

g2 = frame2(100:400, 200:400, :);
for k = numframes-1:-1:1
d2(:, :, k) = imabsdiff(g2(:, :, k), g2(:, :, k+1)); % compute frame differences using imabsdiff
end

for k = numframes-1:-1:1
thresh2(:,:,k) = imbinarize(d2(:,:,k),graythresh(d2(:,:,k)));
end

centroids2 = zeros(numframes, 2);
for k = numframes-1:-1:1
```

```
s2 = regionprops(logical(thresh2(:, :, k)), 'area', 'centroid');
area_vector2 = [s2.Area];
[tmp2, idx2] = max(area_vector2);
centroids2(k, :) = s2(idx2(1)).Centroid;
x2(k) = centroids2(k,1);
y2(k) = centroids2(k,2);
end

xavg2 = mean(x2,2);
yavg2 = mean(y2,2);
smoothx2 = smoothdata(x2,'gaussian',20);
smoothy2 = smoothdata(y2,'gaussian',20);

% camera 3
%
% for i = 1:num3
%     frame3 = rgb2gray(vidFrames3_3(:,:,:,i)); % convert color video into gray scale
%     imshow(frame3(150:350,240:500))
% end

for k = numframes:-1:1
frame3(:, :, k) = rgb2gray(vidFrames3_3(:, :, :, k)); % convert each frame to grayscale using rgb2
end

g3 = frame3(150:350, 240:500, :);
for k = numframes-1:-1:1
d3(:, :, k) = imabsdiff(g3(:, :, k), g3(:, :, k+1)); % compute frame differences using imabsdiff
end

for k = numframes-1:-1:1
thresh3(:,:,k) = imbinarize(d3(:,:,k),graythresh(d3(:,:,k)));
end

centroids3 = zeros(numframes, 2);
for k = numframes-1:-1:1
s3 = regionprops(logical(thresh3(:, :, k)), 'area', 'centroid');
area_vector3 = [s3.Area];
[tmp3, idx3] = max(area_vector3);
centroids3(k, :) = s3(idx3(1)).Centroid;
x3(k) = centroids3(k,1);
y3(k) = centroids3(k,2);
end

xavg3 = mean(x3,2);
yavg3 = mean(y3,2);
smoothx3 = smoothdata(x3,'gaussian',20);
smoothy3 = smoothdata(y3,'gaussian',20);

frame = 1:236;

% plot the xy position

figure(1);
subplot(3,1,1), plot(frame, x1,'r', frame, y1,'b')
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, x2,'r', frame, y2,'b')
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, x3,'r', frame, y3,'b')
```

```
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');

% plot the smooth-xy position (more look like simple harmonic motion)
figure(2);
subplot(3,1,1), plot(frame, smoothx1,'r', frame, smoothy1,'b', 'Linewidth',[2])
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, smoothx2,'r', frame, smoothy2,'b', 'Linewidth',[2])
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, smoothx3,'r', frame, smoothy3,'b', 'Linewidth',[2])
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');

X = [x1-repmat(xavg1,1,236);
    y1-repmat(yavg2,1,236);
    x2-repmat(xavg2,1,236);
    y2-repmat(yavg2,1,236);
    x3-repmat(xavg3,1,236);
    y3-repmat(yavg3,1,236)]; % data matrix
[m n] = size(X);
[u s v] = svd(X/sqrt(n-1)); % svd
lambda = diag(s).^2; % diagonal matrix with eigenvalues of XX^T
Y = u' * X; % principal component basis

X2 = u(:,1)*s(1,1)*v(:,1)';

% plot principal component
figure(3);
plot(lambda/sum(lambda),'ro'), title('Principal Components'); xlabel('variable'); ylabel('singular

% plot Y
figure(4);
subplot(3,1,1), plot(frame, Y(1,:),'r', frame, Y(2,:),'b')
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, Y(3,:),'r', frame, Y(4,:),'b')
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, Y(5,:),'r', frame, Y(6,:),'b')
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');

% plot 1mode
figure(5);
subplot(3,1,1), plot(frame, X2(1,:),'r', frame, X2(2,:),'b')
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, X2(3,:),'r', frame, X2(4,:),'b')
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, X2(5,:),'r', frame, X2(6,:),'b')
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');


clear all; close all; clc;

%%horizontal displacement and rotation

load cam1_4.mat % load the video
num1 = size(vidFrames1_4,4);
load cam2_4.mat
num2 = size(vidFrames2_4,4);
load cam3_4.mat
num3 = size(vidFrames3_4,4);
numframes = min([num1 num2 num3]);
```

```
% camera 1

% for i = 1:num1
%     frame1 = rgb2gray(vidFrames1_4(:,:,:,i)); % convert color video into gray scale
%     imshow(frame1(250:430,300:450)) % see if the cropped one looks good
% end

for k = numframes:-1:1
frame1(:, :, k) = rgb2gray(vidFrames1_4(:, :, :, k)); % convert each frame to grayscale using rgb2
end

g1 = frame1(250:430,300:450,:);
for k = numframes-1:-1:1
d1(:, :, k) = imabsdiff(g1(:, :, k), g1(:, :, k+1)); % compute frame differences using imabsdiff
end

for k = numframes-1:-1:1
thresh1(:,:,k) = imbinarize(d1(:,:,k),graythresh(d1(:,:,k)));
end

centroids1 = zeros(numframes, 2);
for k = numframes-1:-1:1
s1 = regionprops(logical(thresh1(:, :, k)), 'area', 'centroid');
area_vector1 = [s1.Area];
[tmp1, idx1] = max(area_vector1);
centroids1(k, :) = s1(idx1(1)).Centroid;
x1(k) = centroids1(k,1);
y1(k) = centroids1(k,2);
end

xavg1 = mean(x1,2);
yavg1 = mean(y1,2);
smoothx1 = smoothdata(x1,'gaussian',20);
smoothy1 = smoothdata(y1,'gaussian',20);

% camera 2

% for i = 1:num2
%     frame2 = rgb2gray(vidFrames2_4(:,:,:,i)); % convert color video into gray scale
%     imshow(frame2(100:400,200:400))
% end

for k = numframes:-1:1
frame2(:, :, k) = rgb2gray(vidFrames2_4(:, :, :, k)); % convert each frame to grayscale using rgb2
end
g2 = frame2(100:400, 200:400, :);

for k = numframes-1:-1:1
d2(:, :, k) = imabsdiff(g2(:, :, k), g2(:, :, k+1)); % compute frame differences using imabsdiff
end

for k = numframes-1:-1:1
thresh2(:,:,k) = imbinarize(d2(:,:,k),graythresh(d2(:,:,k)));
end

centroids2 = zeros(numframes, 2);
for k = numframes-1:-1:1
s2 = regionprops(logical(thresh2(:, :, k)), 'area', 'centroid');
```

```
area_vector2 = [s2.Area];
[tmp2, idx2] = max(area_vector2);
centroids2(k, :) = s2(idx2(1)).Centroid;
x2(k) = centroids2(k,1);
y2(k) = centroids2(k,2);
end

xavg2 = mean(x2,2);
yavg2 = mean(y2,2);
smoothx2 = smoothdata(x2,'gaussian',20);
smoothy2 = smoothdata(y2,'gaussian',20);

% camera 3
%
% for i = 1:num3
%     frame3 = rgb2gray(vidFrames3_4(:,:,:,i)); % convert color video into gray scale
%     imshow(frame3(150:300,240:500))
% end

for k = numframes:-1:1
frame3(:, :, k) = rgb2gray(vidFrames3_4(:, :, :, k)); % convert each frame to grayscale using rgb2
end
g3 = frame3(150:300, 240:500, :);

for k = numframes-1:-1:1
d3(:, :, k) = imabsdiff(g3(:, :, k), g3(:, :, k+1)); % compute frame differences using imabsdiff
end

for k = numframes-1:-1:1
thresh3(:,:,k) = imbinarize(d3(:,:,k),graythresh(d3(:,:,k)));
end

centroids3 = zeros(numframes, 2);
for k = numframes-1:-1:1
s3 = regionprops(logical(thresh3(:, :, k)), 'area', 'centroid');
area_vector3 = [s3.Area];
[tmp3, idx3] = max(area_vector3);
centroids3(k, :) = s3(idx3(1)).Centroid;
x3(k) = centroids3(k,1);
y3(k) = centroids3(k,2);
end

xavg3 = mean(x3,2);
yavg3 = mean(y3,2);
smoothx3 = smoothdata(x3,'gaussian',20);
smoothy3 = smoothdata(y3,'gaussian',20);

frame = 1:391;

% plot the xy position
figure(1);
subplot(3,1,1), plot(frame, x1,'r', frame, y1,'b')
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, x2,'r', frame, y2,'b')
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, x3,'r', frame, y3,'b')
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
```

```
% plot the smooth-xy position (more look like simple harmonic motion)
figure(2);
subplot(3,1,1), plot(frame, smoothx1,'r', frame, smoothy1,'b', 'Linewidth',[2])
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, smoothx2,'r', frame, smoothy2,'b', 'Linewidth',[2])
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, smoothx3,'r', frame, smoothy3,'b', 'Linewidth',[2])
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');

X = [x1-repmat(xavg1,1,391);
    y1-repmat(yavg2,1,391);
    x2-repmat(xavg2,1,391);
    y2-repmat(yavg2,1,391);
    x3-repmat(xavg3,1,391);
    y3-repmat(yavg3,1,391)]; % data matrix
[m n] = size(X);

[u s v] = svd(X/sqrt(n-1)); % svd
lambda = diag(s).^2; % diagonal matrix with eigenvalues of XX^T
Y = u' * X; % principal component basis

X2 = u(:,1)*s(1,1)*v(:,1)';

figure(3);
plot(lambda/sum(lambda),'ro'), title('Principal Components'); xlabel('variable'); ylabel('singular

figure(4);
subplot(3,1,1), plot(frame, Y(1,:),'r', frame, Y(2,:),'b')
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, Y(3,:),'r', frame, Y(4,:),'b')
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, Y(5,:),'r', frame, Y(6,:),'b')
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');

% plot 1mode
figure(5);
subplot(3,1,1), plot(frame, X2(1,:),'r', frame, X2(2,:),'b')
title('camera1'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,2), plot(frame, X2(3,:),'r', frame, X2(4,:),'b')
title('camera2'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
subplot(3,1,3), plot(frame, X2(5,:),'r', frame, X2(6,:),'b')
title('camera3'); xlabel('Frame'); ylabel('Position'); legend('X', 'Y');
```