Amath 482 Homework 2 Report
PCA and a Spring-Mass System

Colin Kong

February 24th, 2021

Abstract: This assignment started with some movie files created from different cameras. Each camera was set in a different position and they were shooting for videos of a spring-mass system.

Section I: Introduction and Overview

Introduction: The experiments are an attempt to illustrate various aspects of the PCA, its practical usefulness, and the effects of noise on the PCA algorithms. We are showing the experiments in four different tests of 'Ideal Case', 'Noise Case', 'Horizontal Displacement' and 'Horizontal Displacement and Rotation'.

Overview:

I started with loading the videos from the three cameras, and then, in order to narrow down the whole image of the video down to the targeted part of the mass and the spring, I applied a Shannon filter. After filtering, I transformed the colored image to the grayscale image. I calculated for the region of the position of the mass and after that, I applied the PCA algorithm to do the analysis for the videos.

Section II: Theoretical Background

Shannon Filter:

This is just a step function - it takes the values 0 or 1 - so it is just a sharp cut off in frequency space that removes all frequencies above some threshold. Shannon filter is based on the Nyquist–Shannon sampling theorem is a theorem in the field of signal processing which serves as a fundamental bridge between continuous-time signals and discrete-time signals. It establishes a sufficient condition for a sample rate that permits a discrete sequence of samples to capture all the information from a continuous-time signal of finite bandwidth.

SVD (Singular Value Decomposition):

The reduced SVD is not the standard definition of the SVD used in the literature. What is typically done is to construct a matrix U from $\widehat{U}$ by adding an addition m−n column that are orthonormal to the already existing set $\widehat{U}$. In this way, the matrixUbecomes a squarem×mmatrix, and in order to make the decomposition work, and additional m−n row of zeros is also added to the matrix $\widehat{\sum}$, resulting in the matrix Σ. This leads to the singular value decomposition of the matrix A: A = UΣV^*,where U ∈ $\mathbb{R}^{m*n}$ and V ∈ $\mathbb{R}^{m*n}$ are unitary matrices, and Σ ∈ $\mathbb{R}^{m*n}$ is diagonal. Geometrically, the SVD is doing the following. Multiplying by V^* on the left rotates the hypersphere to align the v_n

vectors with the axes. Then we multiply on the left by a diagonal matrix which stretches in each direction. Then we multiply on the left by U to rotate the hyper ellipse to the proper orientation.
To compute SVD, we have:

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

Multiplying on the right by V gives

$$A*AV = V\Sigma^2,$$

so that the columns of V are eigenvectors of A∗A with eigenvalues given by the square of the singular values. Similarly,

$$AA^* = U\Sigma^2 U^*.$$

and so, multiplying on the right by U gives the eigenvalue problem

$$AA^* * U = U\Sigma^2$$

which can be used to solve for U.

PCA (Principle Component Analysis):
Let us now imagine we have multiple vectors. For example, imagine there were 10 assignments and each vector contains grades from the same 20 randomly chosen students. We can put each of those row vectors into a matrix:

$$X = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

We can compute all the variances and covariances between the rows of X with one matrix multiplication:

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

You should notice that C_X is a square symmetric matrix. Unsurprisingly, it is called the covariance matrix. The goal of principal component analysis is to find a new set of coordinates (a change of basis) so that the variables are now uncorrelated. That will mean that each variable contains completely new information, i.e. no redundancies. It would also be nice to know which variables have the largest variance because these contain the most important information about our data. Therefore, we want to diagonalize this matrix so that all off-diagonal elements (covariances) are zero:

$$C_X = V\Lambda V^{-1}.$$

The basis of eigenvectors contained in V are called the principal components. They are uncorrelated since they are orthogonal. Why? Since C_X is a symmetric matrix, its

eigenvalues are real, and the corresponding eigenvectors are orthogonal. The diagonal entries of Λ, the eigenvalues of C_X, are the variances of these new variables. The connection to the SVD for PCA starts with the SVD of a matrix A is connected to the eigenvalue decomposition of AA^T. To account for the $\frac{1}{n-1}$ factor, consider

$$\text{A} = \frac{1}{\sqrt{n-1}}X$$

Then,

$$C_X = \frac{1}{n-1}XX^T = \text{A}A^T.$$

Then, from the following of SVD, we have that C_X = AA^T = U$\Sigma^2 U^T$, where 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. Recall that this factor of $\sqrt{n-1}$ is exactly what we scaled the singular values by to get the right units. Recall that we used a change of basis to work in the basis of the principal components. To do this you multiply by $U^{-1} = U^T$. The data in the new coordinates is

$$Y = U^T X.$$

The covariance of Y is

$$C_Y = \frac{1}{n-1}YY^T = \frac{1}{n-1}U^T XX^T U = U^T \text{A}A^T U = U^T \text{U}\Sigma^2 U^T \text{U} = \Sigma^2$$

Since the off-diagonal elements of Σ are zero, it follows that the variables in Y are uncorrelated.

Spring-Mass System:

A system of masses connected by springs is a classical system with several degrees of freedom. For example, a system consisting of two masses and three springs has two degrees of freedom. This means that its configuration can be described by two generalized coordinates, which can be chosen to be the displacements of the first and second mass from the equilibrium position. The motion of the connected masses is described by two differential equations of second order. In the simplest case we can ignore the forces of friction and air resistance and consider only the elastic force that obeys Hooke's law.

Section III: Algorithm implementation and Development

To record the spring-mass system, we are setting for four tests:

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. MATLAB files: camN_1.mat where N= 1,2,3.

2. Test 2: Noisy Case. Repeat the ideal case experiment, but this time introduce camera

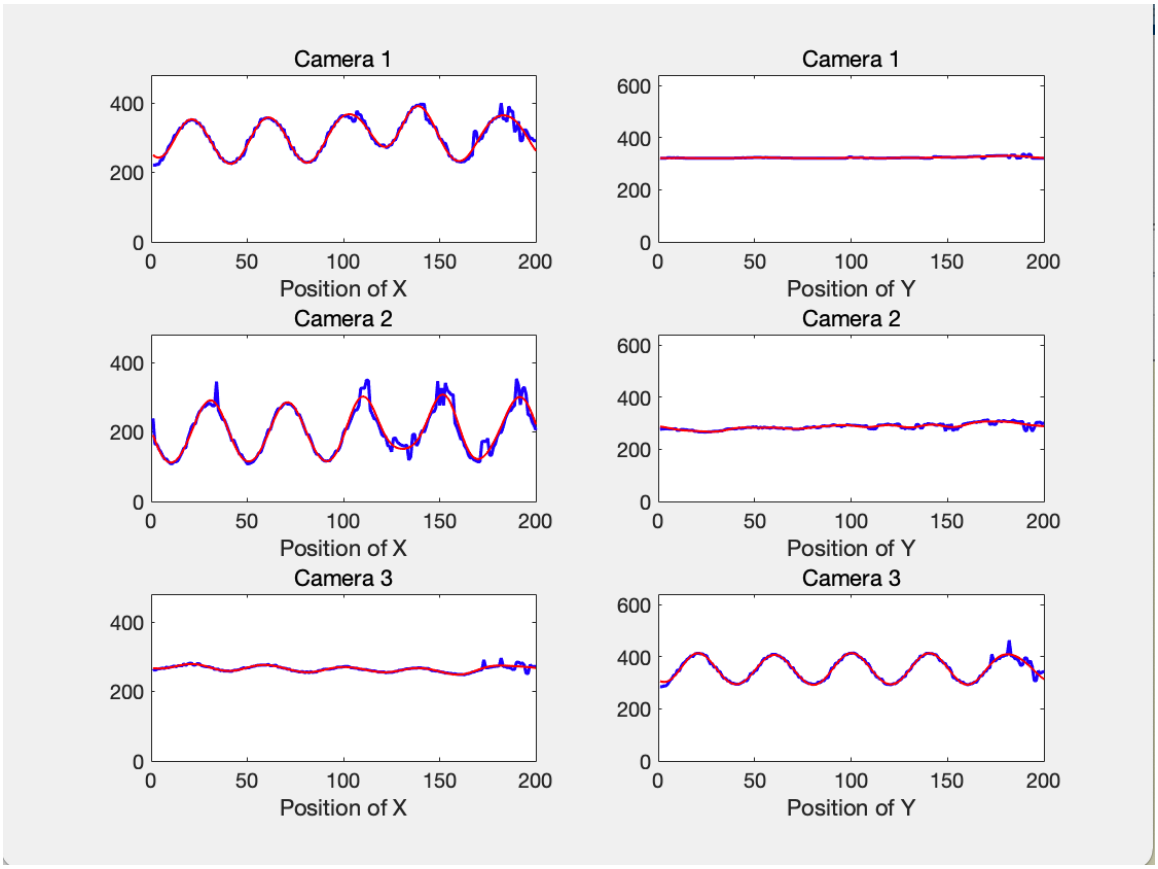
shakes into the video recording. This should make it more difficult to extract the simple harmonic motion. But, if the shake isn't too bad, the dynamics will still be extracted with the PCA algorithms. MATLAB files: camN_2.mat where N= 1,2,3.

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. See what the PCA tells us about the system. MATLAB files: camN_3.mat where N= 1,2,3.

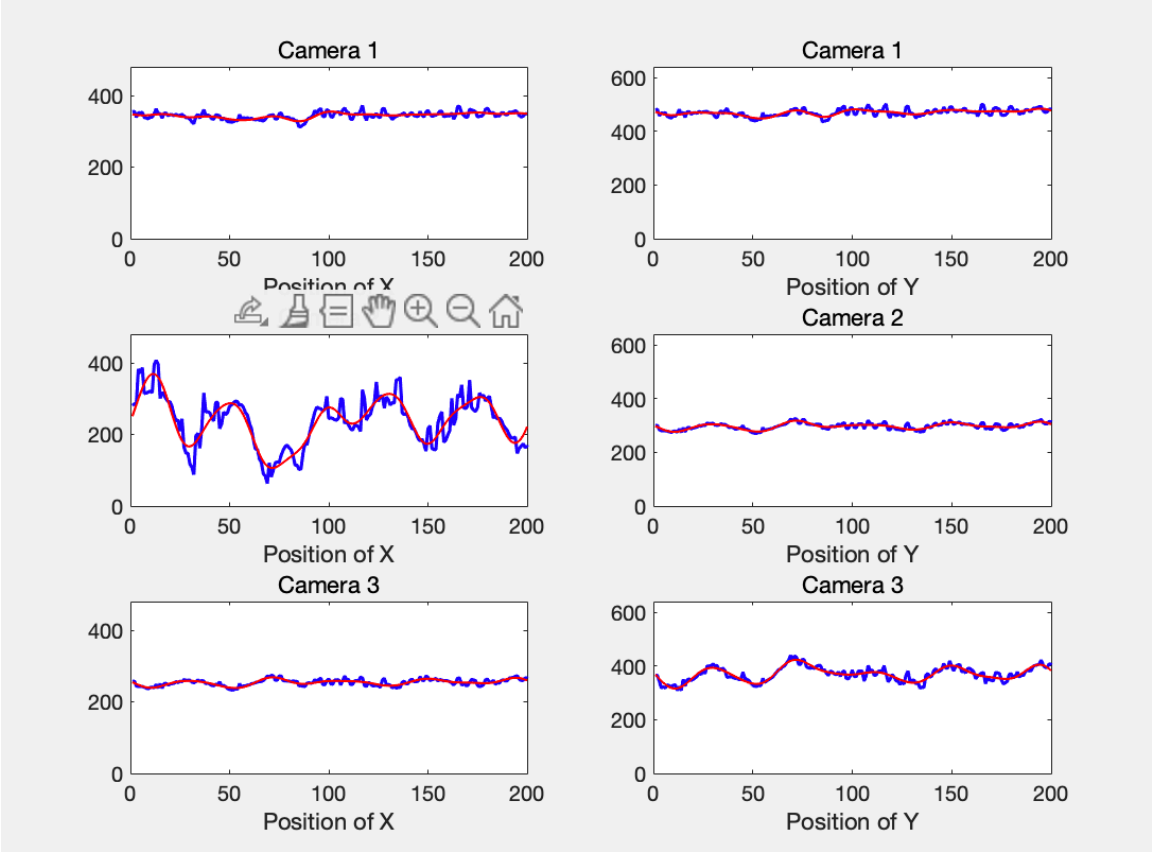
4. Test 4: Horizontal Displacement and Rotation. In this case the mass is released off-center and rotates so as to produce rotation in the x−y plane and motion in the z direction. See what the PCA tells us about the system. MATLAB files: camN_4.mat where N= 1,2,3

Section IV: Computational Results

1. **Displacement of the mass in the z direction in test 1**



2. **Displacement of the mass in the z direction in test 2**



Section V: Summary and Conclusion

In this project, I used 12 pieces of videos took by three cameras for four different tests to explore the various effect of PCA and its practical usefulness and the effects of noise on the PCA algorithms. However, I failed to finish the test 3 and 4.

Appendix A: MATLAB Functions Used

1. **svd(): [U, S, V] = svd(A)** -- where [U, S, V] corresponds to U, Σ *and* V respectively. This command in Matlab calculates the SVD.
2. **rgb2gray()** -- converts the true color image RGB to the grayscale image.
3. **fft(x)** – computes the discrete Fourier transform (DFT) of X using a fast Fourier transform (FFT) algorithm.
4. **X = ifft(Y)** computes the inverse discrete Fourier transform of Y using a fast Fourier transform algorithm

Appendix B: MATLAB Codes

```
clear all;close all;clc;

load('cam1_1.mat');
load('cam2_1.mat');
```

```
load('cam3_1.mat');
[height1_1, width1_1, rgb1_1, numframes1_1] = size(vidFrames1_1);
[height2_1, width2_1, rgb2_1, numframes2_1] = size(vidFrames2_1);
[height3_1, width3_1, rgb3_1, numframes3_1] = size(vidFrames3_1);
%for j=1:num_frames
%    X=vidFrames1_1(:,:,:,j);
%    imshow(X); drawnow
%end
%filter = zeros(480,640);
%filter() = 1;
%% Test 1
X1_1 = [];Y1_1 = [];
X2_1 = [];Y2_1 = [];
X3_1 = [];Y3_1 = [];
for j = 1:numframes1_1
    X = vidFrames1_1(:,:,:,j);
    X_gray = double(rgb2gray(X));
    X_gray(:,1:320) = 0;
    X_gray(:,380:end) = 0;
    X_gray(1:200,:) = 0;
   % X_modified = X_gray .* filter;
    [M,I] = max(X_gray(:));
    [x1_1,y1_1] = ind2sub(size(X_gray),I);
    X1_1 = [X1_1, x1_1];
    Y1_1 = [Y1_1, y1_1];
end
for j = 1:numframes2_1
    X = vidFrames2_1(:,:,:,j);
    X_gray = double(rgb2gray(X));
    X_gray(:,1:260) = 0;
    X_gray(:,320:end) = 0;

    [M,I] = max(X_gray(:));
    [x2_1,y2_1] = ind2sub(size(X_gray),I);
    X2_1 = [X2_1, x2_1];
    Y2_1 = [Y2_1, y2_1];
end
for j = 1:numframes3_1
    X = vidFrames3_1(:,:,:,j);
    X_gray = double(rgb2gray(X));
    X_gray(:,1:250) = 0;
```

```
    X_gray(310:end,:) = 0;
    X_gray(:,1:260) = 0;
    [M,I] = max(X_gray(:));
    [x3_1,y3_1] = ind2sub(size(X_gray),I);
    X3_1 = [X3_1, x3_1];
    Y3_1 = [Y3_1, y3_1];
end
[min, I] = min(X1_1(1:50));
X1_1 = X1_1(I:I+200);Y1_1 = Y1_1(I:I+200);
X2_1 = X2_1(I:I+200);Y2_1 = Y2_1(I:I+200);
X3_1 = X3_1(I:I+200);Y3_1 = Y3_1(I:I+200);
X1_1fft = fft(X1_1);Y1_1fft = fft(Y1_1);
X2_1fft = fft(X2_1);Y2_1fft = fft(Y2_1);
X3_1fft = fft(X3_1);Y3_1fft = fft(Y3_1);
X1_1fft(10:end-10) = 0;Y1_1fft(10:end-10) = 0;
X2_1fft(10:end-10) = 0;Y2_1fft(10:end-10) = 0;
X3_1fft(10:end-10) = 0;Y3_1fft(10:end-10) = 0;
X1_1ifft = abs(ifft(X1_1fft));Y1_1ifft = abs(ifft(Y1_1fft));
X2_1ifft = abs(ifft(X2_1fft));Y2_1ifft = abs(ifft(Y2_1fft));
X3_1ifft = abs(ifft(X3_1fft));Y3_1ifft = abs(ifft(Y3_1fft));
figure(1)
subplot(3,2,1)
plot(X1_1,'b','Linewidth',1.5); hold on;
plot(X1_1ifft,'r','Linewidth',1);
axis([0 200 0 480]);
xlabel("Time(Frames)");xlabel("Position of X");
title("Camera 1")
subplot(3,2,2)
plot(Y1_1,'b','Linewidth',1.5); hold on;
plot(Y1_1ifft,'r','Linewidth',1);
axis([0 200 0 640]);
xlabel("Time(Frames)");xlabel("Position of Y");
title("Camera 1")
subplot(3,2,3)
plot(X2_1,'b','Linewidth',1.5); hold on;
plot(X2_1ifft,'r','Linewidth',1);
axis([0 200 0 480]);
xlabel("Time(Frames)");xlabel("Position of X");
title("Camera 2")
subplot(3,2,4)
plot(Y2_1,'b','Linewidth',1.5); hold on;
```

```
plot(Y2_1ifft,'r','Linewidth',1);
axis([0 200 0 640]);
xlabel("Time(Frames)");xlabel("Position of Y");
title("Camera 2")
subplot(3,2,5)
plot(X3_1,'b','Linewidth',1.5); hold on;
plot(X3_1ifft,'r','Linewidth',1);
axis([0 200 0 480]);
xlabel("Time(Frames)");xlabel("Position of X");
title("Camera 3")
subplot(3,2,6)
plot(Y3_1,'b','Linewidth',1.5); hold on;
plot(Y3_1ifft,'r','Linewidth',1);
axis([0 200 0 640]);
xlabel("Time(Frames)");xlabel("Position of Y");
title("Camera 3")

%% Test 2
load('cam1_2.mat');
load('cam2_2.mat');
load('cam3_2.mat');
[height1_2, width1_2, rgb1_2, numframes1_2] = size(vidFrames1_2);
[height2_2, width2_2, rgb2_2, numframes2_2] = size(vidFrames2_2);
[height3_2, width3_2, rgb3_2, numframes3_2] = size(vidFrames3_2);
X1_2 = [];Y1_2 = [];
X2_2 = [];Y2_2 = [];
X3_2 = [];Y3_2 = [];
for j = 1:numframes1_2
    X = vidFrames1_2(:,:,:,j);
    X_gray = double(rgb2gray(X));
    X_gray(:,1:450) = 0;
    X_gray(:,500:end) = 0;
    X_gray(1:313,:) = 0;
   % X_modified = X_gray .* filter;
    [M,I] = max(X_gray(:));
    [x1_2,y1_2] = ind2sub(size(X_gray),I);
    X1_2 = [X1_2, x1_2];
    Y1_2 = [Y1_2, y1_2];
end
for j = 1:numframes2_2
    X = vidFrames2_2(:,:,:,j);
```

```
    X_gray = double(rgb2gray(X));
    X_gray(:,1:260) = 0;
    X_gray(:,320:end) = 0;
    [M,I] = max(X_gray(:));
    [x2_2,y2_2] = ind2sub(size(X_gray),I);
    X2_2 = [X2_2, x2_2];
    Y2_2 = [Y2_2, y2_2];
end
for j = 1:numframes3_2
    X = vidFrames3_2(:,:,:,j);
    X_gray = double(rgb2gray(X));
    X_gray(:,1:250) = 0;
    X_gray(310:end,:) = 0;
    X_gray(:,1:260) = 0;
    [M,I] = max(X_gray(:));
    [x3_2,y3_2] = ind2sub(size(X_gray),I);
    X3_2 = [X3_2, x3_2];
    Y3_2 = [Y3_2, y3_2];
end
[max, I] = max(X1_2(1:50));
X1_2 = X1_2(I:I+200);Y1_2 = Y1_2(I:I+200);
X2_2 = X2_2(I:I+200);Y2_2 = Y2_2(I:I+200);
X3_2 = X3_2(I:I+200);Y3_2 = Y3_2(I:I+200);
Xsvd = [X1_2;Y1_2;X2_2;Y2_2;X3_2;Y3_2];
[U,S,V] = svd(Xsvd);
Xsvd = U(:,1:2)*S(1:2,1:2)*V(:,1:2)';
X1_2 = Xsvd(1,:);Y1_2 = Xsvd(2,:);
X2_2 = Xsvd(3,:);Y2_2 = Xsvd(4,:);
X3_2 = Xsvd(5,:);Y3_2 = Xsvd(6,:);
X1_2fft = fft(X1_2);Y1_2fft = fft(Y1_2);
X2_2fft = fft(X2_2);Y2_2fft = fft(Y2_2);
X3_2fft = fft(X3_2);Y3_2fft = fft(Y3_2);
X1_2fft(10:end-10) = 0;Y1_2fft(10:end-10) = 0;
X2_2fft(10:end-10) = 0;Y2_2fft(10:end-10) = 0;
X3_2fft(10:end-10) = 0;Y3_2fft(10:end-10) = 0;
X1_2ifft = abs(ifft(X1_2fft));Y1_2ifft = abs(ifft(Y1_2fft));
X2_2ifft = abs(ifft(X2_2fft));Y2_2ifft = abs(ifft(Y2_2fft));
X3_2ifft = abs(ifft(X3_2fft));Y3_2ifft = abs(ifft(Y3_2fft));
Xisvd = [X1_2ifft;Y1_2ifft;X2_2ifft;Y2_2ifft;X3_2ifft;Y3_2ifft];
[Ui,Si,Vi] = svd(Xisvd);
Xisvd = Ui(:,1:2)*Si(1:2,1:2)*Vi(:,1:2)';
```

```
X1_2ifft = Xisvd(1,:);Y1_2ifft = Xisvd(2,:);
X2_2ifft = Xisvd(3,:);Y2_2ifft = Xisvd(4,:);
X3_2ifft = Xisvd(5,:);Y3_2ifft = Xisvd(6,:);
figure(2)
subplot(3,2,1)
plot(X1_2,'b','Linewidth',1.5); hold on;
plot(X1_2ifft,'r','Linewidth',1);
axis([0 200 0 480]);
xlabel("Time(Frames)");xlabel("Position of X");
title("Camera 1")
subplot(3,2,2)
plot(Y1_2,'b','Linewidth',1.5); hold on;
plot(Y1_2ifft,'r','Linewidth',1);
axis([0 200 0 640]);
xlabel("Time(Frames)");xlabel("Position of Y");
title("Camera 1")
subplot(3,2,3)
plot(X2_2,'b','Linewidth',1.5); hold on;
plot(X2_2ifft,'r','Linewidth',1);
axis([0 200 0 480]);
xlabel("Time(Frames)");xlabel("Position of X");
title("Camera 2")
subplot(3,2,4)
plot(Y2_2,'b','Linewidth',1.5); hold on;
plot(Y2_2ifft,'r','Linewidth',1);
axis([0 200 0 640]);
xlabel("Time(Frames)");xlabel("Position of Y");
title("Camera 2")
subplot(3,2,5)
plot(X3_2,'b','Linewidth',1.5); hold on;
plot(X3_2ifft,'r','Linewidth',1);
axis([0 200 0 480]);
xlabel("Time(Frames)");xlabel("Position of X");
title("Camera 3")
subplot(3,2,6)
plot(Y3_2,'b','Linewidth',1.5); hold on;
plot(Y3_2ifft,'r','Linewidth',1);
axis([0 200 0 640]);
xlabel("Time(Frames)");xlabel("Position of Y");
title("Camera 3")
```