Title: PCA and a Spring-Mass System

Author: Wendy Jiang

Abstract: There are four sets of camaras recording files, where each set corresponded to an oscillation scenario. Each set contains three video files to record from three different positions. The cameras observed the same movement of an object from different angles. We would use PCA (Principal Component Analysis) on four tasks to convert three sets of x and y coordinates (six variables at n points in time) into six principal components. Then we plot and analyze the energy of each component. Comparison of those data under different circumstance is the main task of this assignment.

Section I. Introduction and Overview

The different videos of four oscillatory situations which contains noise, horizontal displacement, and rotation behavior. This object motion is a spring-mass system motion. For each case, we could get a matrix contains 6 rows (3 sets of x and y coordinates). A singular value decomposition (SVD) is a way of factoring matrices, but it turns out to be very useful for applications. In this assignment, SVD factorization of a matrix into many constitutive components. SVD is potentially the most powerful and versatile tool that can be used in data analysis. Principle component analysis (PCA) is an application of SVD. Through the process of PCA, we could capture how much energy each principal component gets. By understanding the meaning of SVD, we can see how many of principal component dimensions. This is very significant to estimate if we can accurately reduce the dimension of this system. Since this system is a spring-mass system, the object moving up and down. In different case, there would get different number of principal components after we remove redundant and noise, some videos only need one, however, some need multiple.

Section II. Theoretical Background

1. To scale a vector, we can set a special matrix $A = \begin{bmatrix} \alpha & 0 \\ 0 & \alpha \end{bmatrix}$. For some α, if $\alpha > 1$ then this matrix just stretches vectors by a factor of α, and if $0 < \alpha < 1$ it compresses them by this vector.
   We can label the principle semiaxes of the ellipse resulting from applying the matrix to the unit circle as $\sigma_1 u_1$ and $\sigma_2 u_2$, where σ_1 and σ_2 are the length of the vectors and u_1 and u_2 are unit vectors that point in the directions of the axes. Then, there are some unit vectors v_1 and v_2 such that
   $$Av_1 = \sigma_1 u_1, \qquad Av_2 = \sigma_2 u_2 \quad \ldots(1)$$
   Which means $AV = U\Sigma$. $\ldots(2)$
   Where V is a unitary matrix with v_1 and v_2 as its columns, U is unitary matrix with u_1 and u_2 as its columns, and Σ is a diagonal matrix with σ_j on its diagonal. Since V is unitary, we can isolate for A to get the **Singular Value Decomposition**:
   $$A = U\Sigma V^* \quad \ldots(3)$$

2. The diagonal values of σ are nonnegative and ordered from largest to smallest. We can also compute the SVD with:

$$A^T A = (U\Sigma V^*)^T (U\Sigma V^*) = (V\Sigma \mathrm{U})(U\Sigma V^*) = V\Sigma V^* \quad \ldots (4)$$

$$A^T A = (U\Sigma V^*)(U\Sigma V^*)^T = (U\Sigma \mathrm{V})(V\Sigma U^*) = U\Sigma U^* \quad \ldots (5)$$

$$A^T AV = V\Sigma^2 \quad \ldots (6)$$
$$AA^T U = U\Sigma^2 \quad \ldots (7)$$

Comparison the SVD and Eigenvalue Decomposition, they both are about finding bases, but the eigenvalue decomposition finds a single basis while the SVD uses two different ones. The eigenvectors of A are orthogonal then the singular values are just the absolute values of the eigenvalues and the singular vectors are eigenvectors. The SVD enables for every matrix of any size to be diagonal if the proper base for the domain and the range are used. This is extremely important because the real-world data does not always come in the form of square matrix.

3. The method that uses SVD to produce low-rank approximations of a data set is called principal component analysis. This method allows one to quantify low dimensional dynamics without any knowledge of underlying behavior. We can put each of those row vectors into a matrix: $X = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$. Then we can compute all the variance and covariance between the rows of X with one matrix multiplication:

$$C_x = \frac{1}{n-1} XX^T. \quad \ldots (8)$$

We should note that C_x is a square symmetric matrix and it is called the covariance matrix. This covariance matrix is a key to understanding redundancies in data: large values correspond to redundancy. But large diagonal terms also correspond to large variances, which suggest strong fluctuations in specific variables, and help identify important components. Large variances are thus dynamic of interest, while small variances are non-interesting. Thus, we want to diagonalize the C_x matrix so that all off-diagonal elements (covariances) are zeros:

$$C_x = V\Lambda V^{-1} \quad \ldots (9).$$

The basis of eigenvectors contained in V are called the principal components. They are uncorrelated since they are orthogonal. 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 variances of these new variables.

4. The data in the new coordinates is $Y = U^T X$. $\ldots (10)$
The covariance of Y is

$$C_Y = \frac{1}{n-1} YY^T = \frac{1}{n-1} U^T XX^T U = U^T AA^T U = U^T U\Sigma^2 U^T U = \Sigma^2 \quad \ldots (11)$$

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

5. Spring-Mass system: a mass is hanging from a spring and the mass bobs up and down. Simply note the solution of the vertical displacement of the mass:
$$z(t) = Acos(\omega t + \varphi) \quad \dots(12)$$
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.

6. In general, PCA suggest that we are expanding our solution in another orthonormal basis, where we can diagonalize the system. We are normally given a function $f(x,t)$, which we must expand in a different basis representation such that
$$f(x,t) = \Sigma_{j+1}^{N} a_j(t)\phi_j(x) \quad \dots(13)$$
From this expansion, we can consider the energy in each principal component direction, which are the singular values σ_j. One note when we do SVA and PCA is that we must subtract the mean for each row x_j. PCA also assumes linearity, and that larger variances are more important. PCA may not always produce optimal result.

7. We measure the energy contained in the rank-N approximation by:
$$energy_N = \frac{{\sigma_1}^2 + {\sigma_2}^2 + \cdots {\sigma_N}^2}{{\sigma_1}^2 + {\sigma_2}^2 + \cdots + {\sigma_r}^2} = \frac{||X_N||_F^2}{||X||_F^2} \quad \dots(14)$$
The above formular actually corresponds to the physical energy for these types of problems: it has a connection with the Frobenius norm; we will use the word energy a lot while talking about the SVD, even when we are taking the SVD of an imagine.

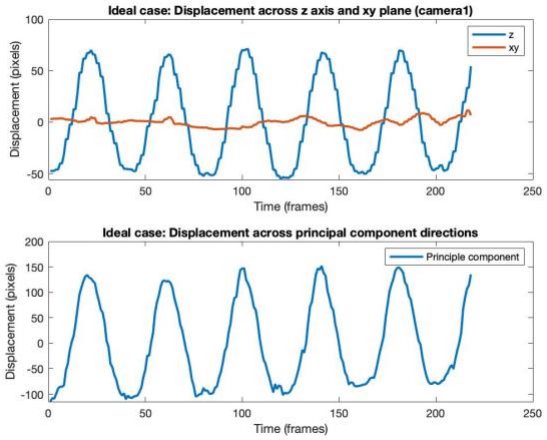
Algorithm Development

We have four different case here, procedures for all four case are similar. Here I clarify the process of one case. We need to load all three videos from different positions at the very beginning. Then use **size** command to catch the number of frames and the whole video's width and height. Then we create a loop to go through each frame and convert the colorful frame to grayscale by **rgb2gray** and **double** command in order to prepare use the function below. We can see there always a most bright spot at the top of the can, so we could track the object by this spot after we convert the frame into grayscale. We always want to adjust the threshold to avoid the NaN in our data. Therefore, we set the threshold to filter out the value and keep the brightest spot's value here. Then, we use the **find** function to locate the value in the matrix (the indices of these points) and use **ind2sub** function to get the exact coordinates of all these bright spot points. Then we average the coordinates and add it to the data entry. As a result, we could collect data of each frames. Repeat this process of each of three camaras. Since there are three set of data, we should make each set of data corresponds to each other, hence, we use `[M,I] = min(data3(1:20,2))` to make sure they started at the same time. To be more convenience for plot we identify the shortest video's length and cut their data into same length. We can get a three set of x and y coordinates after the process above. We combine these 3 sets x and y coordinates into a 6 rows matrix. This matrix will be used into SVD.

With the data matrix, we then subtracted the mean of each row from the data to center the data. We use the SVD of our data matrix and divided by $\sqrt{n-1}$. We extracted the value of the

diagonal matrix S and squared them to get the variances for each principal component. The last step is to plot the normalized variances which divided by the sum value, in order to analyze which principal component were significant. And finally, we plot the projections of data on the significant principal component orthonormal bases to reconstruct our observed data.

Computational Result



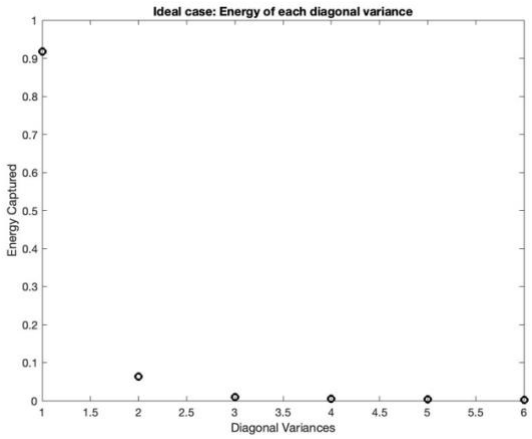
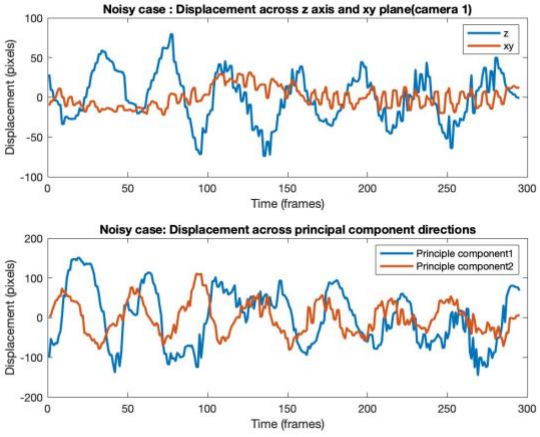


Figure 1. plot of ideal case of vertical displacement vs. x-y plane (left)
Figure 2. plot of the variances of principal component (energy) (right)



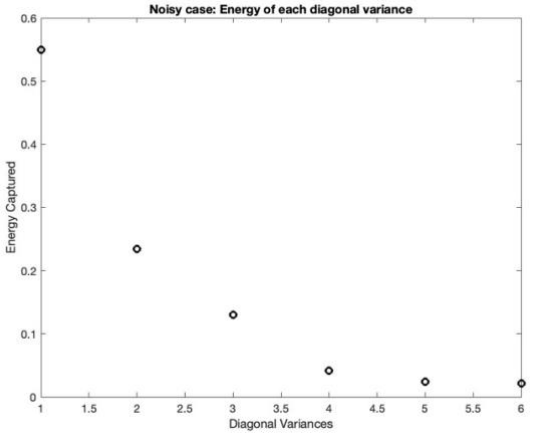
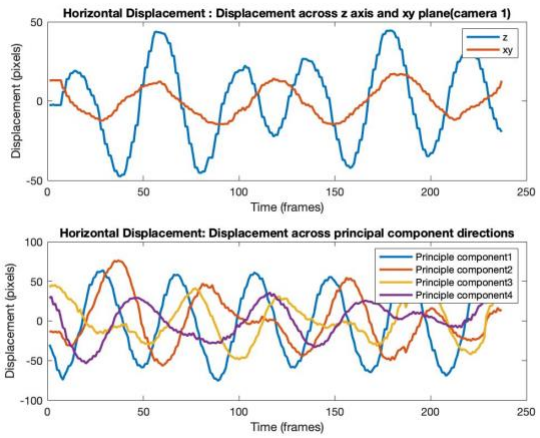


Figure 3. plot of noisy case of vertical displacement vs. x-y plane (left)
Figure 4. plot of the variances of principal component (energy) (right)



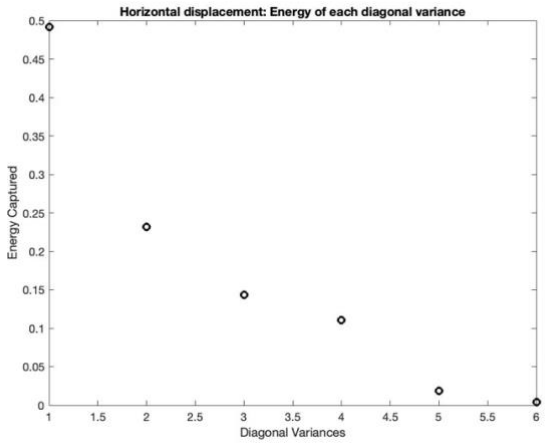
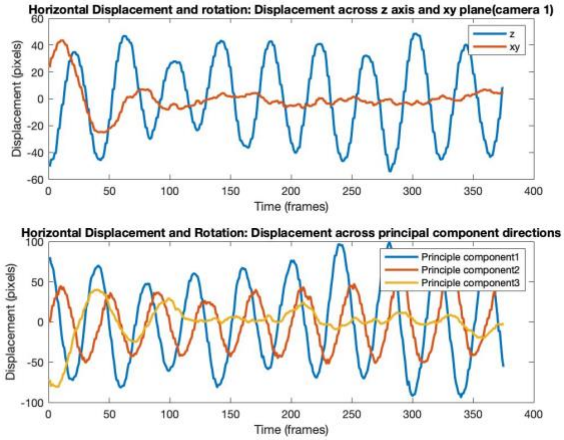


Figure 5. plot of horizontal displacement case vertical displacement vs new basis(left)
Figure 6. plot of the variances of principal component (energy) (right)



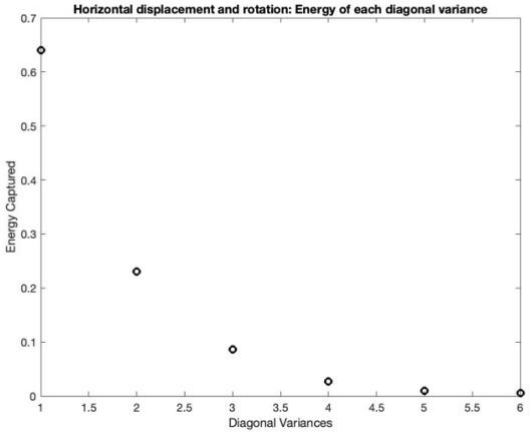


Figure 7. plot of horizontal displacement and Rotation case vertical displacement vs new basis(left)
Figure 8. plot of the variances of principal component (energy) (right)

How many principal components it needs of each case? We shouldn't expect only one nonzero singular value, but we should expect that one is significantly larger than all the others, representing the dominant up-down motion of the mass.
To understand how many lines in the plot of the displacement across Principal component directions we should get in each case, we should see the plot of variance of principal component(energy). Here we assume if we can capture energy significantly over 0, it counts in PC.

- **Case 1. Ideal case**

  By Figure1. The plot to show the movement of the can of camera 1. The x-y plane is almost a straight line, and the trajectory of the moving can very close to the cosine plot, which means here's only one dominant movement. By Figure 2., we can find that only had one principal component which corresponded to 91% energy. There is a big gap between the first two points in Figure2, the rest points have really low value of energy capturing, as a result, it should only need one principal component in this ideal case. In other words, the transformation of the first principal component is more like the accurate data. For our projection plot, it accurately fit the ideal system.

- **Case 2. Noisy Case**

Looking the Figure 4., there are a bigger gap between first two points and also a gap between the second point and the third point. The gap means the differences of variance. We can assume two Principal component it needs, which means there are two points are significantly high. We can see the first point's value is clearly smaller than the first point's value of the Figure 4. Although this case is similar to the ideal case, the noise actually influences the variance. In Figure 3's top plot, obviously, the noise exist as oscillates. For the projection to the PC basis, the noise oscillates as well, there are more fluctuate based on the cosine curve.

- **Case3. Horizontal Displacement**

In Figure 6, there's four significant points' value, so we assume it needs four principal components which can capture high amount of energy. These four variances' gap are smaller than the gap in other cases. The energy of the case 3's diagonal variance is lower than the first and second case. By the top plot of the Figure 5, the x-y plane curve is more like the cosine, however, the up and down movement(z) has been affected by the horizontal displacement. In the below plot of the Figure5, we could see multiple direction exist by PC curves, which also adapt to both vertical and horizontal displacement in the video and the curves have significant differences than case 1 and 2.

- **Case4. Horizontal Displacement and Rotation**

In Figure 8, there seem to be 3 significant principal component that have high energy captured. This corresponds to the observation; it has both oscillatory motion and spinning motion. Hence, we could assume PCA captures multi-dimension displacement, no matter the rotates or horizontal displacement. The PC directions fit the displacement we get.

Summary and Conclusion

PCA is used in exploratory data analysis and for making predictive models. It is commonly used for dimensionality reduction by projecting each data point onto only the first few principal components to obtain lower-dimensional data while preserving as much of the data's variation as possible. In this assignment, we are able to track the oscillating object for different scenarios and perform PCA on datasets to see the significancy and principal components. We enable to plot the energy captured by specific nodes, and the original z projected vs. the basis sets of the principal component. And by this plot, we could analyze how many principal components the case need. By the plot of displacement across the PC direction, we enable to see the movement which is corresponds to the video movement.

Appendix A.

- `diag(X)`: Returns a column vector of the diagonal values of a matrix. It can generate the variance number by SVD.
- `find(X)`: Returns a vector of non-zero indices in a matrix. It enables to locate the bright spots of each images. Also, it corresponded to moving object.
- `y = filter(b,a,X)`: filters the input data x using a rational transfer function defined by the numerator and denominator coefficients b and a.

- `[row,col] = ind2sub(sz,ind)`: returns the arrays row and col containing the equivalent row and column subscripts corresponding to the linear indices ind for a matrix of size sz.
- `I = rgb2gray(RGB)`: converts the true color image RGB to the grayscale image I. The rgb2gray function converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance.
- `M = mean(A)` returns the mean of the elements of A along the first array dimension whose size does not equal 1.

Appendix B.

```
clear all; close all; clc;

%% Test1: ideal case
% input
load('cam1_1.mat'); load('cam2_1.mat');load('cam3_1.mat');
numFrames1a = size(vidFrames1_1,4);
numFrames2a = size(vidFrames2_1,4);
numFrames3a = size(vidFrames3_1,4);
[height1 width1 rgb1 num_frames1] = size(vidFrames1_1);
[height2 width2 rgb2 num_frames2] = size(vidFrames2_1);
[height3 width3 rgb3 num_frames3] = size(vidFrames3_1);
%%
% filter
width = 50;
filter = zeros(480,640);
filter(300-2.6*width:1:300+2.6*width, 350-width:1:350+width) = 1;

data1 = []; % inital blank matrix setting
for j = 1:numFrames1a % run each frame
    X = vidFrames1_1(:,:,:,j);
    Xabw = rgb2gray(X); % convert colorful to gray
    Xabw2 = double(Xabw); % convert unit to double
    %imshow(X);drawnow
    Xf = Xabw2.*filter; % use filter to data
    thresh = Xf > 250; % select the bright spot
    indeces = find(thresh);
    [Y,X] = ind2sub(size(thresh),indeces); %locate
    data1 = [data1; mean(X), mean(Y)]; % get one set coordinates
end

%%

width = 50;
filter = zeros(480,640);
filter(250-3*width:1:250+3*width, 290-1.3*width:1:290+1.3*width) = 1;
data2 = [];
for j = 1:numFrames2a
    X = vidFrames2_1(:,:,:,j);
    Xabw = rgb2gray(X);
    Xabw2 = double(Xabw);
    Xf = Xabw2.*filter;
    thresh = Xf> 250;
```

```
    indeces = find(thresh);
    [Y,X] = ind2sub(size(thresh),indeces);
    data2 = [data2; mean(X), mean(Y)];
end
%%

width = 50;
filter = zeros(480,640);
filter(250-width:1:250+2*width, 360-2.5*width:1:360+2.5*width) = 1;
data3 = [];
for j = 1:numFrames3a
    X = vidFrames3_1(:,:,:,j);
    Xabw = rgb2gray(X);
    Xabw2 = double(Xabw);
    Xf = Xabw2.*filter;
    thresh = Xf > 247;
    indeces = find(thresh);
    [Y,X] = ind2sub(size(thresh),indeces);
    data3 = [data3; mean(X), mean(Y)];
end
%% let all video data corresponds
[M,I] = min(data1(1:20,2));
data1 = data1(I:end,:);
%
[M,I] = min(data2(1:20,2));
data2 = data2(I:end,:);
%
[M,I] = min(data3(1:20,2));
data3 = data3(I:end,:);
%% convert to the same length
data3 = data3(1:length(data1),:);
data2 = data2(1:length(data1),:);

%% combine the data to one matrix and use svd
all = [data1 data2 data3]';
[m,n] = size(all);
mn = mean(all,2); % find mean value
all = all - repmat(mn,1,n); % minus mean value
[U,S,V] = svd(all/sqrt(n-1)); % svd
lambda = diag(S).^2; % get lambda
Y = U'*all; % the variance
sig = diag(S);

%%
figure(1)
plot(1:6,lambda/sum(lambda),'ko','linewidth',2);
title('Ideal case: Energy of each diagonal variance');
xlabel('Diagonal Variances'); ylabel('Energy Captured');


%%
figure(2)
subplot(2,1,1)
plot(1:218,all(2,:),1:218,all(1,:),'linewidth',2);
xlabel('Time (frames)'); ylabel('Displacement (pixels)');
title('Ideal case: Displacement across z axis and xy plane (camera1)');
```

```
legend('z','xy');

subplot(2,1,2)
plot(1:218,Y(1,:),'linewidth',2);
xlabel('Time (frames)'); ylabel('Displacement (pixels)');
title('Ideal case: Displacement across principal component directions');
legend('Principle component');



%%
clear all; close all; clc;
%% Test2: noisy case
load('cam1_2.mat'); load('cam2_2.mat');load('cam3_2.mat');
numFrames1b = size(vidFrames1_2,4);
numFrames2b = size(vidFrames2_2,4);
numFrames3b = size(vidFrames3_2,4);

width = 50;
filter = zeros(480,640);
filter(300 - 2.6*width:1:300+2.6*width, 350-width:1:350+2*width) = 1;

data1 = [];
for j = 1:numFrames1b
    X = vidFrames1_2(:,:,:,j);
    %imshow(X);drawnow
    Xabw = rgb2gray(X);
    Xabw2 = double(Xabw);
    Xf = Xabw2.*filter;
    ind = find(Xf > 250);
    [Y,X] = ind2sub(size(Xf > 250),ind);
    data1 = [data1; mean(X),mean(Y)];
end
%%
width = 50;
filter = zeros(480,640);
filter(250-3*width:1:250+3*width, 290-1.3*width:1:290+1.3*width) = 1;
data2 = [];
for j = 1:numFrames2b
    X = vidFrames2_2(:,:,:,j);
    Xabw = rgb2gray(X);
    Xabw2 = double(Xabw);
    Xf = Xabw2.*filter;
    ind2 = find(Xf > 237);
    [Y,X] = ind2sub(size(Xf > 237),ind2);
    data2 = [data2; mean(X), mean(Y)];
end

%%
width = 50;
filter = zeros(480,640);
filter(250-width:1:250+2*width, 360-2.5*width:1:360+2.5*width) = 1;
data3 = [];
for j = 1:numFrames3b
    X = vidFrames3_2(:,:,:,j);
```

```
    Xabw = rgb2gray(X);
    Xabw2 = double(Xabw);
    Xf = Xabw2.*filter;
    ind3 = find(Xf > 245);
    [Y,X] = ind2sub(size(Xf > 245),ind3);
    data3 = [data3; mean(X), mean(Y)];
end
%%
[M,I] = min(data1(1:20,2));
data1 = data1(I:end,:);

[M,I] = min(data2(1:20,2));
data2 = data2(I:end,:);


[M,I] = min(data3(1:20,2));
data3 = data3(I:end,:);

%%
 data2 = data2(1:length(data1),:);

 data3 = data3(1:length(data1),:);
%
alldata = [data1 data2 data3]';
%%
[m2,n2] = size(alldata);
mn2 = mean(alldata,2);
alldata = alldata - repmat(mn2,1,n2);
[U,S,V] = svd(alldata/sqrt(n2-1));
lambda2 = diag(S).^2;
Y = U'*alldata;

%%
figure(3)
plot(1:6,lambda2/sum(lambda2),'ko','linewidth',2);
title('Noisy case: Energy of each diagonal variance');
xlabel('Diagonal Variances'); ylabel('Energy Captured');
%%
figure(4)
subplot(2,1,1)
plot(1:287, alldata(2,:),1:287,alldata(1,:),'linewidth',2);
xlabel('Time (frames)'); ylabel('Displacement (pixels)');
title('Noisy case : Displacement across z axis and xy plane(camera 1)');
legend('z','xy');

% subplot(3,1,2)
% plot(1:314, alldata(4,:),1:314,alldata(3,:),'linewidth',2);
% xlabel('Time (frames)'); ylabel('Displacement (pixels)');
% title('Noisy case : camera 2');
% legend('z','xy');
%
% subplot(3,1,3)
% plot(1:314, alldata(6,:),1:314,alldata(5,:),'linewidth',2);
% xlabel('Time (frames)'); ylabel('Displacement (pixels)');
% title('Noisy case : camera 3');
```

```
% legend('z','xy');

subplot(2,1,2)
plot(1:287,Y(1,:),1:287,Y(2,:),'linewidth',2);
xlabel('Time (frames)'); ylabel('Displacement (pixels)');
title('Noisy case: Displacement across principal component directions');
legend('Principle component1','Principle component2');


%%
clear all; close all; clc;
%% Test3: Horizontal Displacement
load('cam1_3.mat'); load('cam2_3.mat');load('cam3_3.mat');
numFrames1b = size(vidFrames1_3,4);
numFrames2b = size(vidFrames2_3,4);
numFrames3b = size(vidFrames3_3,4);

width = 50;
filter = zeros(480,640);
filter(300 - 2.6*width:1:300+2.6*width, 350-width:1:350+2*width) = 1;

data1 = [];
for j = 1:numFrames1b
    X = vidFrames1_3(:,:,:,j);
    Xabw = rgb2gray(X);
    Xabw2 = double(Xabw);
    Xf = Xabw2.*filter;
    ind = find(Xf > 250);
    [Y,X] = ind2sub(size(Xf > 250),ind);
    data1 = [data1; mean(X),mean(Y)];
end
%%
width = 50;
filter = zeros(480,640);
filter(250-3*width:1:250+3*width, 290-1.3*width:1:290+1.3*width) = 1;
data2 = [];
for j = 1:numFrames2b
    X = vidFrames2_3(:,:,:,j);
    Xabw = rgb2gray(X);
    Xabw2 = double(Xabw);
    Xf = Xabw2.*filter;
    ind2 = find(Xf > 250);
    [Y,X] = ind2sub(size(Xf > 250),ind2);
    data2 = [data2; mean(X), mean(Y)];
end

%%
width = 50;
filter = zeros(480,640);
filter(250-width:1:250+2*width, 360-2.5*width:1:360+2.5*width) = 1;
data3 = [];
for j = 1:numFrames3b
    X = vidFrames3_3(:,:,:,j);
    Xabw = rgb2gray(X);
    Xabw2 = double(Xabw);
    Xf = Xabw2.*filter;
```

```
    ind3 = find(Xf > 247);
    [Y,X] = ind2sub(size(Xf > 247),ind3);
    data3 = [data3; mean(X), mean(Y)];
end

%%
[M,I] = min(data1(1:20,2));
data1 = data1(I:end,:);

%%
[M,I] = min(data2(1:20,2));
data2 = data2(I:end,:);

%%
[M,I] = min(data3(1:20,2));
data3 = data3(I:end,:);
%%
data1 = data1(1:length(data3),:);
data2 = data2(1:length(data3),:);
alldata = [data1 data2 data3]';
%%
[m3,n3] = size(alldata);
mn3 = mean(alldata,2);
alldata = alldata - repmat(mn3,1,n3);

[U,S,V] = svd(alldata/sqrt(n3-1));
lambda3 = diag(S).^2;
Y = U'*alldata;



%%
figure(5)
plot(1:6,lambda3/sum(lambda3),'ko','linewidth',2);
title('Horizontal displacement: Energy of each diagonal variance');
xlabel('Diagonal Variances'); ylabel('Energy Captured');

%%
figure(5)
subplot(2,1,1)
plot(1:237, alldata(2,:),1:237,alldata(1,:),'linewidth',2);
xlabel('Time (frames)'); ylabel('Displacement (pixels)');
title('Horizontal Displacement : Displacement across z axis and xy
plane(camera 1)');
legend('z','xy');

% subplot(3,1,2)
% plot(1:237, alldata(4,:),1:237,alldata(3,:),'linewidth',2);
% xlabel('Time (frames)'); ylabel('Displacement (pixels)');
% title('Horizontal Displacement : camera 2');
% legend('z','xy');
%
% subplot(3,1,3)
% plot(1:237, alldata(6,:),1:237,alldata(5,:),'linewidth',2);
% xlabel('Time (frames)'); ylabel('Displacement (pixels)');
% title('Horizontal displacement : camera 3');
```

```
% legend('z','xy');
subplot(2,1,2)
plot(1:237,Y(1,:),1:237,Y(2,:),1:237,Y(3,:), 'linewidth',2);
xlabel('Time (frames)'); ylabel('Displacement (pixels)');
title('Horizontal Displacement: Displacement across principal component
directions');
legend('Principle component1','Principle component2','Principle component3');


%%
clear all; close all; clc;
%% Test4: Horizontal Displacement and rotation
load('cam1_4.mat'); load('cam2_4.mat');load('cam3_4.mat');
numFrames1b = size(vidFrames1_4,4);
numFrames2b = size(vidFrames2_4,4);
numFrames3b = size(vidFrames3_4,4);

width = 50;
filter = zeros(480,640);
filter(300 - 1.5*width:1:300+3*width, 350-1.5*width:1:350+2*width) = 1;

data1 = [];
for j = 1:numFrames1b
    X = vidFrames1_4(:,:,:,j);%imshow(X);drawnow
    Xabw = rgb2gray(X);
    Xabw2 = double(Xabw);
    Xf = Xabw2.*filter;
    ind = find(Xf > 243);
    [Y,X] = ind2sub(size(Xf > 243),ind);
    data1 = [data1; mean(X),mean(Y)];
end
%%
width = 50;
filter = zeros(480,640);
filter(250-3*width:1:250+3.5*width, 290-2.5*width:1:290+2.7*width) = 1;
data2 = [];
for j = 1:numFrames2b
    X = vidFrames2_4(:,:,:,j);%imshow(X);drawnow
    Xabw = rgb2gray(X);
    Xabw2 = double(Xabw);
    Xf = Xabw2.*filter;
    ind2 = find(Xf > 245);
    [Y,X] = ind2sub(size(Xf > 245),ind2);
    data2 = [data2; mean(X), mean(Y)];
end

%%
width = 50;
filter = zeros(480,640);
filter(250-1.8*width:1:250+2.5*width, 360-2.5*width:1:360+2.8*width) = 1;
data3 = [];
for j = 1:numFrames3b
    X = vidFrames3_4(:,:,:,j);
    Xabw = rgb2gray(X);
    Xabw2 = double(Xabw);
    Xf = Xabw2.*filter;
```

```
    ind3 = find(Xf > 234);
    [Y,X] = ind2sub(size(Xf > 234),ind3);
    data3 = [data3; mean(X), mean(Y)];
end

%%
[M,I] = min(data1(1:20,2));
data1 = data1(I:end,:);

[M,I] = min(data2(1:20,2));
data2 = data2(I:end,:);

[M,I] = min(data3(1:20,2));
data3 = data3(I:end,:);

%%

data1 = data1(1:length(data3),:);
data2 = data2(1:length(data3),:);


alldata = [data1 data2 data3]';
%%
[m4,n4] = size(alldata);
mn4 = mean(alldata,2);
alldata = alldata - repmat(mn4,1,n4);
[U,S,V] = svd(alldata/sqrt(n4-1));
lambda4 = diag(S).^2;
Y = U'*alldata;


%%
figure(7)
plot(1:6,lambda4/sum(lambda4),'ko','linewidth',2);
title('Horizontal displacement and rotation: Energy of each diagonal
variance');
xlabel('Diagonal Variances'); ylabel('Energy Captured');

%%
figure(8)
subplot(2,1,1)
plot(1:375, alldata(2,:),1:375,alldata(1,:),'linewidth',2);
xlabel('Time (frames)'); ylabel('Displacement (pixels)');
title('Horizontal Displacement and rotation: Displacement across z axis and
xy plane(camera 1)');
legend('z','xy');

% subplot(3,1,2)
% plot(1:232, alldata(4,:),1:232,alldata(3,:),'linewidth',2);
% xlabel('Time (frames)'); ylabel('Displacement (pixels)');
% title('Horizontal Displacement : camera 2');
% legend('z','xy');
%
% subplot(3,1,3)
% plot(1:232, alldata(6,:),1:232,alldata(5,:),'linewidth',2);
```

```
% xlabel('Time (frames)'); ylabel('Displacement (pixels)');
% title('Horizontal displacement : camera 3');
% legend('z','xy');
subplot(2,1,2)
plot(1:375,Y(1,:),1:375,Y(2,:),1:375,Y(3,:),'linewidth',2);
xlabel('Time (frames)'); ylabel('Displacement (pixels)');
title('Horizontal Displacement and Rotation: Displacement across principal
component directions');
legend('Principle component1','Principle component2','Principle component3');
```