Homework 3: PCA Analysis on Paint Can Trajectory

Lahari Gorantla
AMATH 482

Abstract

We were given four scenarios with different oscillatory behaviors of a man standing in classroom bouncing a paint can with a flashlight on top by a string. The four different cases- the ideal case, the noisy case, horizontal displacement, and horizontal displacement with rotation were all filmed with three different camera angles each. This oscillatory motion was tracked and then analyzed with PCA to illustrate its usefulness and the effects of noise on the PCA algorithm by creating energy plots and principal component displacement plots.

I. Introduction and Overview

Principal component analysis (PCA) is a technique used to analyze complex or random datasets by reducing the dimensionality and redundancies while retaining as much information as possible. It can be used to describe the behavioral and dynamical data with limited dimensions and variables when governing equations are not known. This analysis can be used to study the dynamics and behaviors of turbulent fluid flow, neural activity, and image processing, to only name a few.

In this lab report, we will perform PCA analysis on four different oscillatory behaviors of a paint can with a flashlight on top. In case 1, the ideal case, the man moves the paint can up and down, denoted as the *Z direction.* In case 2, the noisy case, while the behavior of the man is the same, the video camera is shaking sideways to create substantial noise. In case 3, the horizontal displacement case, there is movement in both the *Z direction* and some movement in the *X-Y plane*, created by releasing the paint can off-center. In case 4, the horizontal displacement and rotation case, there is movement in both the *Z direction* and rotation, created by releasing the paint can off-center. The purpose of this lab report is to assume we don't know the trajectory mechanics of the paint can and its governing equations. We will use the 3 cameras for each case to extract data about the system and experimentally understand the simplified behavior.

For each case, the flashlight on top of the paint can will be tracked to create a matrix with movement in the *Z direction* and the *XY plane* across the length of the number of frames of the video. Then matrix with the trajectory of the flashlight will be analyzed with singular value decomposition (SVD), a step in PCA. In an energy plot, the singular values of the decomposed data display the number of significant principle components and the amount of energy each captures. In a principal component displacement plot, we can see the plot of the motion assumed by the SVD to see the assumed behavior.

II. Theoretical Background

Principal Component Analysis

PCA is a linearity dimensionality reduction method used in many applications to reconstruct the ideal or simplified behavior of a system and its governing equations by taking a complicated dataset and reducing its dimensionality and redundancies. The purpose to reduce linearly correlated variables into linearly uncorrelated variables to simplify the system. The first principal component is the largest as it captures most of the variance in the system. The following principal components capture other uncorrelated variables in the data. Each of the principal component is orthogonal and uncorrelated to the previous one. PCA generally has two steps, first is to normalize the data and the second is to do SVD or do an eigenvalue decomposition of the covariance matrix. For this lab, the SVD method will used, which will work by diagonalizing the covariance matrix.

Singular Value Decomposition

The SVD is a method that produces the principal components of a normalized dataset. It reduces a matrix to its basic constituent parts to simplify further calculations. The decomposition takes the form:

$$A = U\Sigma V^* \qquad (1)$$

Where A is the original matrix and $U \in \mathbb{C}^{m\times m}$ is unitary, $V \in \mathbb{C}^{n\times n}$ is unitary, and $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal. The *m* denotes the number of measurements while the *n* denotes the number of data points. The matrixes work such that V stores information about the shape of the original matrix, Σ stretches the data with its principal semi-axes, and U rotates the data to its axis. This method give unique singular values, σ_j, to display variance within the dataset after reducing the dimensions. To compute the SVD, this is the process used:

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

And:

$$\begin{aligned} AA^T &= (U\Sigma V^*)(U\Sigma V^*)^T \\ &= U\Sigma^2U^* \end{aligned} \qquad (3)$$

Then, multiplying the right-hand side by V and U, produces two eigenvalue problems.

$$A^TAV = V\Sigma^2 \qquad (4)$$
$$AA^TU = U\Sigma^2 \qquad (5)$$

The square root of the eigenvalues of the equations produces the singular values, σ_j. Normalized eigenvectors for these equations give the basis vectors for U and V. MATLAB easily does the calculations with *svd* where U, S, and V correspond to U, Σ, and V, respectively.

Covariance Matrix

The SVD method can diagonalize the covariance matrix, which contains the covariance between the data sets, by using U for unitary transformation.

$$Y = U^{*}X \qquad (6)$$

Where Y denotes the transformed variable and X is the original matrix. The Y matrix produces the principal components projection. The Y matrix displays the reduced number of variables needed to explain data. In addition to the principal component projection data, the covariance matrix gives us a relationship to find the energy encapsulated per principal number. The covariance matrix functions by reducing the system to its low dimension, redundancies (some from the oversampling with 3 video cameras) are removed, and the variances are listed from largest to smallest. Large variances correspond to more redundancies and more variation in the data, and therefore contain the essence of the system. C_Y in the following equation is the covariance matrix.

$$C_Y = \frac{1}{n-1} YY^T \qquad (7)$$

Which can be arranged by using the relationship $Y=U^{*}X$ and $X= U\Sigma V^{*}$ to produce:

$$C_Y = \frac{1}{n-1} \Sigma^2 \qquad (8)$$

The demonstrates the relationship between the covariance of the data and the squared singular values of the system. This can be used in an energy plot, which describes how much of the data is being described by that singular value. The energy can help the user determine exactly which vectors from the Y matrix to use to describe the data.

III. Algorithm Implementation and Development

For each case, 3 of the video cameras are loaded into MATLAB and the number of frames are found. Then, a spatial filter is mad to only focus on the motion of the bright flashlight and white paint can without interference from the wall and white board. To find the dimensions of the spatial filter, the *pixel region* function on video figures is used. After converting the colorful video into a grayscale video, the filter is multiplied so that all regions without the flashlight and paint can combo are zeros. Then, the filtered grayscale video is converted into a binary logical matrix of 0s and 1s. Given a certain threshold value, *imbinarize* converts values above that to 1s and those below that threshold to 0s. Using this function, the flashlight and some bright parts of the can were found per frame. Then by using *find*, all non-zero bright spot values were returned and placed into the original frame via *ind2sub***.** The white spot we are tracking isn't limited to a single region so we want to find a centroid between the locations. Therefore, we take the mean of the x and y indices returned by *ind2sub* per frame. An important thing to note is that the y axis of an image starts from the top and goes to the bottom so every time we plotted, we had to use 480 minus the mean y position we found. Another important thing to note is that every 3rd video given to us displayed a paint can moving "up and down" horizontally. To utilize this video properly, we called this sideways motion the *Z direction* and treated it as such.

However, none of the videos are synced so we manually went in to find the closest minimum trough or maximum peak in the 3 cameras to truncate the video, ensuring it both started at the same time and had the same number of frames. For the 3rd video in each case, we manipulated the axis to make sure that when the paint can was at its maximum stretch, it corresponded to the maximum stretch of the other two videos as well, and appeared as such on the plots.

Then the synced videos are concatenated to create a 6 × number of frames matrix in the X of camera 1, Y of camera 1, X of camera 2, etc. for all 3 cameras. Then to center the data, we subtracted the mean of each row from the data in the corresponding row. Then we used *svd* of our data divided by $\sqrt{n-1}$ to find the U, S, and V matrices. Then we made normalized energy plots by squaring the S for each principal components and then dividing it by the sum of all S squared. To find the principal component projections we multiplied U by the original data. Then by using the energy plots, we decided what principal component projections to plot.

IV. Computational Results

Case 1- Ideal Case: As seen in Figure 01, the 1st principal component captured 96.7% of the energy. The motion was mostly in 1 direction. When the 1st, 2nd, and 3rd principal components were used to reconstruct the motion, the 1st component captured the oscillatory motion of the object with high accuracy. This makes sense because all the movement was in the *Z direction*.

Case 2- Noisy Case: In this case, the camera was shaking in the *XY* plane. As seen in Figure 02, the energies captured were 71.8%, 13.6%, 6.7%, and 4.5% by the 1st, 2nd, 3rd, and 4th principal components, respectively. As seen by both the *XY* and *Z displacement* graphs, there was noise in the *XY plane* that caused the presence of more than just the 1st principal component. PCA1 projected the *Z displacement* while PCA2 and PCA3 projected the *XY plane* noise.

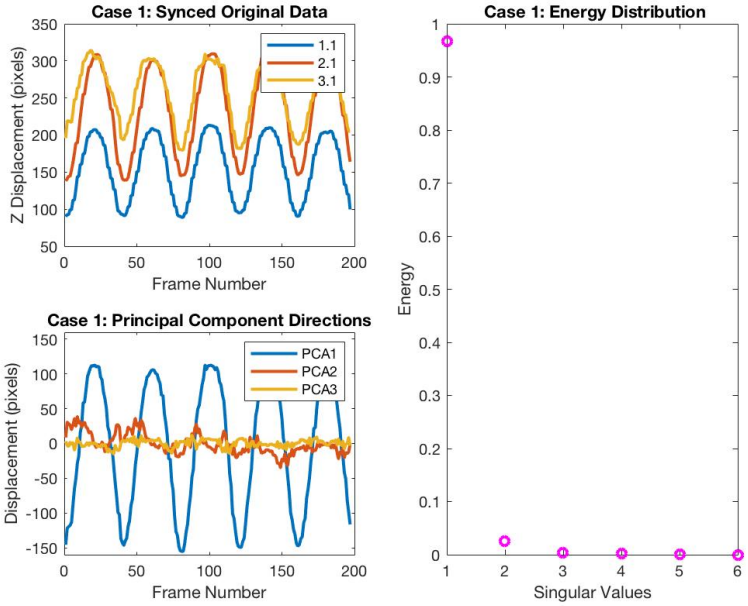

Figure 01. PCA analysis of Case 1.

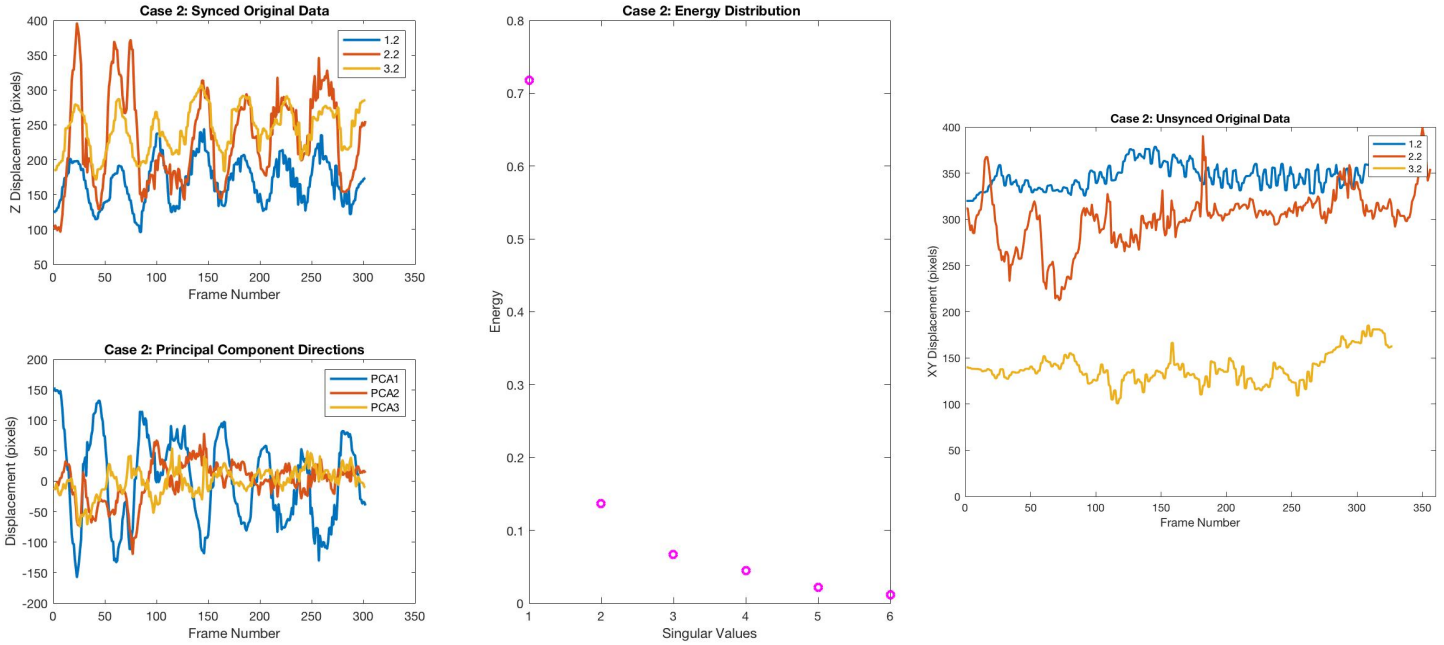

Figure 02. PCA analysis of Case 2 with both XY and Z displacement plots.

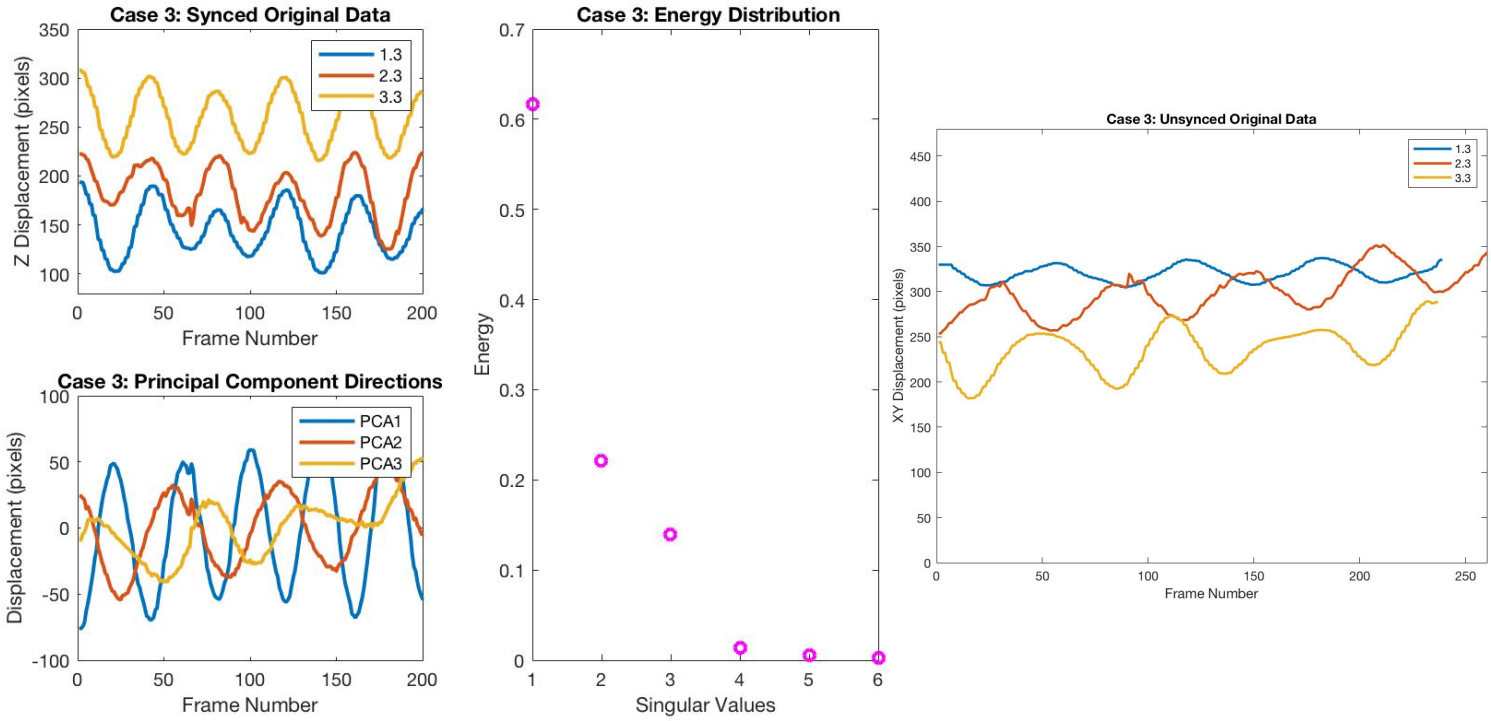

Figure 03. PCA analysis of Case 3 with both XY and Z displacement plots.

Case 3- Horizontal Displacement: As seen in Figure 03, the energies were 61.7%, 22.1%, and 13.9%, respectively for the 1st, 2nd, and 3rd principal components. The PCAs projected in the principal component plots look extremely similar to the original *XY* and *Z displacement* plots.

Case 4- Horizontal Displacement + Rotation: As seen in Figure 04, the energies were 83.5%, 9.4%, and 3.6%, respectively. The PCAs projected in the principal component plots look extremely similar to the original *XY* and *Z displacement* plots.

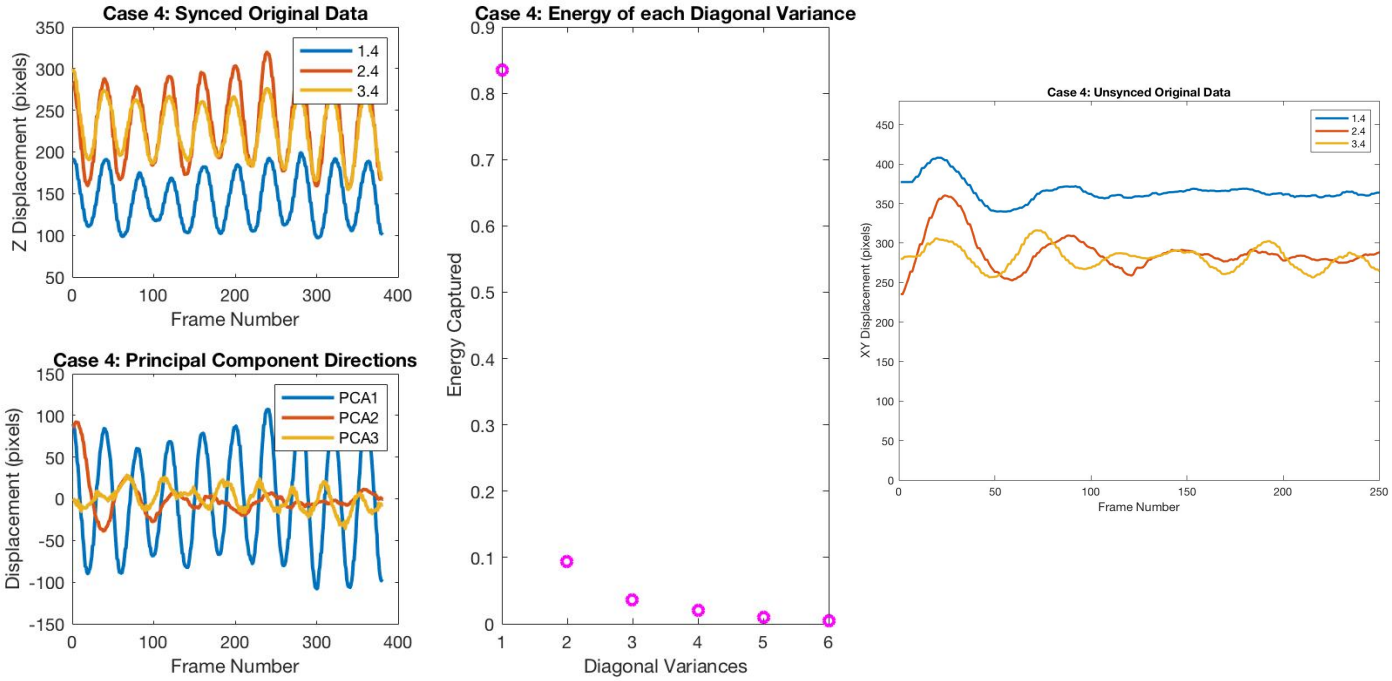

Figure 04. PCA analysis of Case 4 with both XY and Z displacement plots.

V. Summary and Conclusion

PCA is an effective technique for identifying the ideal, simplified behavior of our system by identifying the number of significant, orthonormal principal components. In this lab, it predicted the motion of the flashlight and paint can with high accuracy after reducing the dimensionality and redundancies of the system. The higher the energy of the PCA, the more characteristic it is to the system we are studying. One thing to note is that the noise in case 2 was disruptive and gave us a predicted output that wouldn't have been true to our system.

Appendix A. MATLAB functions used and brief implementation explanation

Rbg2gray(x) – turns a colored image into a greyscale one
Imbinarize(x) – converts values above a threshold into 1s and values below the threshold to 0s
Find(x) – returns indices of non-zero values
Ind2sub(x) – places the indices into the original dimensions of the matrix

Appendix B. MATLAB codes

```
clear all; close all; clc;

%CAM1_1
load('cam1_1.mat');numFrames1 = size(vidFrames1_1,4);
%creates a spatial filter, initializing the mean vectors
filter = zeros(480,640);
filter(200:430,300:400) = 1;
mean_x1 = zeros(1,length(numFrames1));
mean_y1 = zeros(1,length(numFrames1));


set(0, 'DefaultLineLineWidth', 2);

for j = 1:numFrames1

    X = vidFrames1_1(:,:,:,j);
    filter_uint8 = uint8(filter); %converts filter to an uint8 type
    gray_vid1 = rgb2gray(X); %turns to grayscale
    filt_vid1 = gray_vid1.*filter_uint8; %applies the spatial filter

    %thresh = filt_vid1 > 250; %0.97
    %could also binarize it and drop the tolerance
    thresh = imbinarize(filt_vid1,0.97);
    %imshow(thresh)

    %finds all non-zero vectors
    indeces = find(thresh);

    %finds the matrix/vectors
    [Y, X] = ind2sub(size(thresh),indeces);

    %finds the centroid!
    mean_x1(j) = mean(X);
    mean_y1(j) = mean(Y);

end

%invert the mean_y1 because the axis of a picture is upside down
image_y = 480 - mean_y1;

% CAM2_1
load('cam2_1.mat'); numFrames2 = size(vidFrames2_1,4);
filter = zeros(480,640);
filter(100:385,260:340) = 1;
mean_x2 = zeros(1,length(numFrames2));
mean_y2 = zeros(1,length(numFrames2));
for j = 1:numFrames2

    X = vidFrames2_1(:,:,:,j);
    filter_uint8 = uint8(filter);
    gray_vid2 = rgb2gray(X);
    filt_vid2 = gray_vid2.*filter_uint8;

    %thresh = filt_vid2 > 250; %0.97
    %imshow(thresh); drawnow
    thresh = imbinarize(filt_vid2,0.97);
    indeces2 = find(thresh);
    [Y2, X2] = ind2sub(size(thresh),indeces2);
    mean_x2(j) = mean(X2);
    mean_y2(j) = mean(Y2);

end
image_y2 = 480-mean_y2;

% CAM3_1
load('cam3_1.mat'); numFrames3 = size(vidFrames3_1,4);
filter = zeros(480,640);
filter(230:330,270:485) = 1; %manually found the filter
mean_x3 = zeros(1,length(numFrames3));
mean_y3 = zeros(1,length(numFrames3));

for j = 1:numFrames3

    X = vidFrames3_1(:,:,:,j);
```

```
    %imshow(X); drawnow
    filter_uint8 = uint8(filter);
    gray_vid3 = rgb2gray(X);
    filt_vid3 = gray_vid3.*filter_uint8;
    thresh = imbinarize(filt_vid3,0.95);
    %thresh = filt_vid3 > 240; %0.95
    %imshow(thresh); drawnow
    indeces3 = find(thresh);
    [Y3, X3] = ind2sub(size(thresh),indeces3);
     mean_x3(j) = mean(X3);
     mean_y3(j) = mean(Y3);
end
%opposite axis
image_y3 = 480-mean_y3;


plot(1:numFrames1,mean_x1); axis([0 250 0 350]);hold on;
plot(1:numFrames2,mean_x2); hold on; plot(1:numFrames3,480-mean_y3);
legend('1.1','2.1','3.1')
title('Case 1: Unsynced Original Data');
xlabel('Frame Number');
ylabel('XY Displacement (pixels)');




%%
figure(1); %time!
image_x3 = 640-mean_x3;
plot(1:numFrames1,image_y); axis([0 numFrames1 0 350]); hold on;
plot(1:numFrames2,image_y2); hold on; plot(1:numFrames3,image_x3);
legend('1.1','2.1','3.1')

%crop the videos to match
min1 = 30; min2 = 39; min3 = 30; width = 196;
shift_mean1 = mean_y1(min1:min1+width);
shift_mean2 = mean_y2(min2:min2+width);
shift_mean3 = mean_x3(min3:min3+width);

figure(2);
subplot(2,2,1)
plot(1:197,480 - shift_mean1); hold on;
plot(1:197,480 - shift_mean2); hold on;
plot(1:197,640 - shift_mean3); hold on;
legend('1.1','2.1','3.1')
title('Case 1: Synced Original Data');
xlabel('Frame Number');
ylabel('Z Displacement (pixels)');

%,shift_mean2,shift_mean3);
% figure(2); %positions
% subplot(3,1,1); plot(mean_x1, image_y,'.'); axis([0 640 0 480]);
% subplot(3,1,2); plot(mean_x2, image_y2,'.'); axis([0 640 0 480]);
% subplot(3,1,3); plot(mean_x3, image_y3,'.'); axis([0 640 0 480]);

concat = [mean_x1(min1:min1+width);shift_mean1;mean_x2(min2:min2+width);...
    shift_mean2;mean_x3(min3:min3+width);shift_mean3];
%mean weights something less, normalize, average out teh center prosition


test1mat = [concat(1,:)-mean(concat(1,:));concat(2,:)-mean(concat(2,:));...
    concat(3,:)-mean(concat(3,:));concat(4,:)-mean(concat(4,:));...
    concat(5,:)-mean(concat(5,:));concat(6,:)-mean(concat(6,:))];

[u,s,v] = svd(test1mat/sqrt(196));

lambda = diag(s).^2; % produce diagonal variances
%Y= test1mat;
Y = u'*test1mat;
%Y = u'*test1mat;
sig= diag(s);
subplot(2,2,[2 4])
plot (1:6 ,lambda/sum(lambda), 'mo ');
title (" Case 1: Energy Distribution");
xlabel ("Singular Values"); ylabel ("Energy");

subplot(2,2,3);
```

```
%plot(1:381,Y(1,:),1:381,Y(2,:),1:381,Y(3,:),1:381,Y(4,:),1:381,Y(5,:),1:381,Y(6,:));
plot(1:197,Y(1,:),1:197,Y(2,:),1:197,Y(3,:)); axis([0 200 -160 160]);
legend('PCA1','PCA2','PCA3');
title('Case 1: Principal Component Directions');
xlabel('Frame Number');
ylabel('Displacement (pixels)');


clear all; close all; clc;

%CAM1_3
load('cam1_4.mat');numFrames1 = size(vidFrames1_4,4);
%creates a spatial filter, initializing

set(0, 'DefaultLineLineWidth', 2);

filter = zeros(480,640);
filter(225:420,305:467) = 1;
mean_x1 = zeros(1,length(numFrames1));
mean_y1 = zeros(1,length(numFrames1));

for j = 1:numFrames1

    X = vidFrames1_4(:,:,:,j);
    %imshow(X); drawnow

    filter_uint8 = uint8(filter); %converts filter to an uint8 type
    gray_vid1 = rgb2gray(X); %turns to grayscale
    filt_vid1 = gray_vid1.*filter_uint8; %applies the spatial filter

    %thresh = filt_vid1 > 250; %0.97
    %could also binarize it and drop the tolerance
    thresh = imbinarize(filt_vid1,0.95);
    %imshow(thresh)

    %finds all non-zero vectors
    indeces = find(thresh);

    %finds the matrix/vectors
    [Y, X] = ind2sub(size(thresh),indeces);

    %finds the centroid!
    mean_x1(j) = mean(X);
    mean_y1(j) = mean(Y);

end

    %invert the mean_y1 because the axis of a picture is upside down
    image_y = 480 - mean_y1;

%plot(mean_x1,image_y,'.'); axis([0 640 0 480])

%CAM1_3
load('cam2_4.mat');numFrames2 = size(vidFrames2_4,4);
%creates a spatial filter, initializing

filter = zeros(480,640);
filter(104:375,200:430) = 1;
mean_x2 = zeros(1,length(numFrames2));
mean_y2 = zeros(1,length(numFrames2));

for j = 1:numFrames2

    X = vidFrames2_4(:,:,:,j);
    %imshow(X); drawnow

    filter_uint8 = uint8(filter); %converts filter to an uint8 type
    gray_vid2 = rgb2gray(X); %turns to grayscale
    filt_vid2 = gray_vid2.*filter_uint8; %applies the spatial filter

    %thresh = filt_vid1 > 250; %0.97
    %could also binarize it and drop the tolerance
    thresh = imbinarize(filt_vid2,0.95);
    %imshow(thresh)

    %finds all non-zero vectors
    indeces = find(thresh);
```

```
    %finds the matrix/vectors
    [Y, X] = ind2sub(size(thresh),indeces);

    %finds the centroid!
    mean_x2(j) = mean(X);
    mean_y2(j) = mean(Y);


end


%invert the mean_y1 because the axis of a picture is upside down
image_y2 = 480 - mean_y2;

%CAM1_3
load('cam3_4.mat');numFrames3 = size(vidFrames3_4,4);
%creates a spatial filter, initializing

filter = zeros(480,640);
filter(137:283,300:511) = 1;
mean_x3 = zeros(1,length(numFrames3));
mean_y3 = zeros(1,length(numFrames3));

for j = 1:numFrames3

    X = vidFrames3_4(:,:,:,j);
    %imshow(X); drawnow

    filter_uint8 = uint8(filter); %converts filter to an uint8 type
    gray_vid3 = rgb2gray(X); %turns to grayscale
    filt_vid3 = gray_vid3.*filter_uint8; %applies the spatial filter

    %thresh = filt_vid1 > 250; %0.97
    %could also binarize it and drop the tolerance
    thresh = imbinarize(filt_vid3,0.92);
    %imshow(thresh)

    %finds all non-zero vectors
    indeces = find(thresh);

    %finds the matrix/vectors
    [Y, X] = ind2sub(size(thresh),indeces);

    %finds the centroid!
    mean_x3(j) = mean(X);
    mean_y3(j) = mean(Y);

end

%invert the mean_y1 because the axis of a picture is upside

image_y3 = 480 - mean_y3;
image_x3 = 640 - mean_x3;




plot(1:numFrames1,mean_x1); axis([0 250 0 480]);hold on;
plot(1:numFrames2,mean_x2); hold on; plot(1:numFrames3,480-mean_y3);
legend('1.4','2.4','3.4')
title('Case 4: Unsynced Original Data');
xlabel('Frame Number');
ylabel('XY Displacement (pixels)');



%%
figure();
plot(1:numFrames1,image_y); axis([0 360 0 450]); hold on;
plot(1:numFrames2,image_y2); hold on; plot(1:numFrames3,image_x3);
title('Z');
legend('1_4','2_4','3_4')

figure();
plot(1:numFrames1,mean_x1); axis([0 360 0 450]); hold on;
plot(1:numFrames2,mean_x2); hold on; plot(1:numFrames3,image_y3);
title('XY');
legend('1_4','2_4','3_4')
```

```
min1 = 12; min2 = 21; min3 = 14; width = 380;
% 380
% 384
% 380
shift_mean1 = mean_y1(min1:min1+width);
shift_mean2 = mean_y2(min2:min2+width);
shift_mean3 = mean_x3(min3:min3+width);

figure();
subplot(2,2,1);
plot(1:381,480 - shift_mean1); hold on;
plot(1:381,480 - shift_mean2); hold on;
plot(1:381,640 - shift_mean3); hold on;
legend('1.4','2.4','3.4')
title('Case 4: Synced Original Data');
xlabel('Frame Number');
ylabel('Z Displacement (pixels)');

concat = [mean_x1(min1:min1+width);shift_mean1;mean_x2(min2:min2+width);...
    shift_mean2;mean_y3(min3:min3+width);shift_mean3];
%mean weights something less, normalize, average out teh center prosition


test1mat = [concat(1,:)-mean(concat(1,:));concat(2,:)-mean(concat(2,:));...
    concat(3,:)-mean(concat(3,:));concat(4,:)-mean(concat(4,:));...
    concat(5,:)-mean(concat(5,:));concat(6,:)-mean(concat(6,:))];

%svd analysis
[u,s,v] = svd(test1mat/sqrt(380));

lambda = diag(s).^2; % produce diagonal variances
%Y= test1mat;
Y = u'*test1mat;
%Y = u'*test1mat;
sig= diag(s);
subplot(2,2,[2 4]);
plot (1:6 ,lambda/sum(lambda), 'mo ');
title (" Case 4: Energy of each Diagonal Variance ");
xlabel (" Diagonal Variances "); ylabel (" Energy Captured ");

subplot(2,2,3);
%plot(1:381,Y(1,:),1:381,Y(2,:),1:381,Y(3,:),1:381,Y(4,:),1:381,Y(5,:),1:381,Y(6,:));
plot(1:381,Y(1,:),1:381,Y(2,:),1:381,Y(3,:));
legend('PCA1','PCA2','PCA3');
legend('PCA1','PCA2','PCA3');
title('Case 4: Principal Component Directions');
xlabel('Frame Number');
ylabel('Displacement (pixels)');

clear all; close all; clc;

%CAM1_3
load('cam1_3.mat');numFrames1 = size(vidFrames1_3,4);
%creates a spatial filter, initializing

set(0, 'DefaultLineLineWidth', 2);

filter = zeros(480,640);
filter(214:427,268:400) = 1;
mean_x1 = zeros(1,length(numFrames1));
mean_y1 = zeros(1,length(numFrames1));

for j = 1:numFrames1

    X = vidFrames1_3(:,:,:,j);
    %imshow(X); drawnow

    filter_uint8 = uint8(filter); %converts filter to an uint8 type
    gray_vid1 = rgb2gray(X); %turns to grayscale
    filt_vid1 = gray_vid1.*filter_uint8; %applies the spatial filter

    %thresh = filt_vid1 > 250; %0.97
    %could also binarize it and drop the tolerance
    thresh = imbinarize(filt_vid1,0.9);
    %imshow(thresh)
```

```
    %finds all non-zero vectors
    indeces = find(thresh);

    %finds the matrix/vectors
    [Y, X] = ind2sub(size(thresh),indeces);

    %finds the centroid!
    mean_x1(j) = mean(X);
    mean_y1(j) = mean(Y);

end
%invert the mean_y1 because the axis of a picture is upside down
image_y = 480 - mean_y1;


plot(mean_x1,image_y,'.'); axis([0 640 0 480])

%CAM1_3
load('cam2_3.mat');numFrames2 = size(vidFrames2_3,4);
%creates a spatial filter, initializing

filter = zeros(480,640);
filter(177:420,211:423) = 1;
mean_x2 = zeros(1,length(numFrames2));
mean_y2 = zeros(1,length(numFrames2));

for j = 1:numFrames2

    X = vidFrames2_3(:,:,:,j);
    %imshow(X); drawnow

    filter_uint8 = uint8(filter); %converts filter to an uint8 type
    gray_vid2 = rgb2gray(X); %turns to grayscale
    filt_vid2 = gray_vid2.*filter_uint8; %applies the spatial filter

    %thresh = filt_vid1 > 250; %0.97
    %could also binarize it and drop the tolerance
    thresh = imbinarize(filt_vid2,0.9);
    %imshow(thresh)

    %finds all non-zero vectors
    indeces = find(thresh);

    %finds the matrix/vectors
    [Y, X] = ind2sub(size(thresh),indeces);

    %finds the centroid!
    mean_x2(j) = mean(X);
    mean_y2(j) = mean(Y);


end


%invert the mean_y1 because the axis of a picture is upside down
image_y2 = 480 - mean_y2;

%CAM1_3
load('cam3_3.mat');numFrames3 = size(vidFrames3_3,4);
%creates a spatial filter, initializing

filter = zeros(480,640);
filter(154:337,245:495) = 1;
mean_x3 = zeros(1,length(numFrames3));
mean_y3 = zeros(1,length(numFrames3));

for j = 1:numFrames3

    X = vidFrames3_3(:,:,:,j);
    %imshow(X); drawnow

    filter_uint8 = uint8(filter); %converts filter to an uint8 type
    gray_vid3 = rgb2gray(X); %turns to grayscale
    filt_vid3 = gray_vid3.*filter_uint8; %applies the spatial filter

    %thresh = filt_vid1 > 250; %0.97
    %could also binarize it and drop the tolerance
    thresh = imbinarize(filt_vid3,0.9);
```

```
    %imshow(thresh)

    %finds all non-zero vectors
    indeces = find(thresh);

    %finds the matrix/vectors
    [Y, X] = ind2sub(size(thresh),indeces);

    %finds the centroid!
    mean_x3(j) = mean(X);
    mean_y3(j) = mean(Y);

end

%invert the mean_y1 because the axis of a picture is upside down
image_y3 = 480 - mean_y3;
image_x3 = 640 - mean_x3;



plot(1:numFrames1,mean_x1); axis([0 260 0 480]);hold on;
plot(1:numFrames2,mean_x2); hold on; plot(1:numFrames3,480-mean_y3);
legend('1.3','2.3','3.3')
title('Case 3: Unsynced Original Data');
xlabel('Frame Number');
ylabel('XY Displacement (pixels)');


clear all; close all; clc;

%CAM2_1
load('cam1_2.mat');numFrames1 = size(vidFrames1_2,4);
%creates a spatial filter, initializing

filter = zeros(480,640);
filter(213:405,296:417) = 1;
mean_x1 = zeros(1,length(numFrames1));
mean_y1 = zeros(1,length(numFrames1));

for j = 1:numFrames1

    X = vidFrames1_2(:,:,:,j);
    %imshow(X); drawnow

    filter_uint8 = uint8(filter); %converts filter to an uint8 type
    gray_vid1 = rgb2gray(X); %turns to grayscale
    filt_vid1 = gray_vid1.*filter_uint8; %applies the spatial filter

    %thresh = filt_vid1 > 250; %0.97
    %could also binarize it and drop the tolerance
    thresh = imbinarize(filt_vid1,0.98);
    %imshow(thresh)

    %finds all non-zero vectors
    indeces = find(thresh);

    %finds the matrix/vectors
    [Y, X] = ind2sub(size(thresh),indeces);

    %finds the centroid!
    mean_x1(j) = mean(X);
    mean_y1(j) = mean(Y);

end

%invert the mean_y1 because the axis of a picture is upside down
image_y = 480 - mean_y1;

%CAM2_2
load('cam2_2.mat');numFrames2 = size(vidFrames2_2,4);
%creates a spatial filter, initializing

filter = zeros(480,640);
filter(63:415,181:430) = 1;
mean_x2 = zeros(1,length(numFrames2));
mean_y2 = zeros(1,length(numFrames2));

for j = 1:numFrames2
```

```
    X = vidFrames2_2(:,:,:,j);
    %imshow(X); drawnow

    filter_uint8 = uint8(filter); %converts filter to an uint8 type
    gray_vid2 = rgb2gray(X); %turns to grayscale
    filt_vid2 = gray_vid2.*filter_uint8; %applies the spatial filter

    %thresh = filt_vid1 > 250; %0.97
    %could also binarize it and drop the tolerance
    thresh = imbinarize(filt_vid2,0.98);
    %imshow(thresh)

    %finds all non-zero vectors
    indeces = find(thresh);

    %finds the matrix/vectors
    [Y, X] = ind2sub(size(thresh),indeces);

    %finds the centroid!
    mean_x2(j) = mean(X);
    mean_y2(j) = mean(Y);


end


%invert the mean_y1 because the axis of a picture is upside down
image_y2 = 480 - mean_y2;



%CAM3_2
load('cam3_2.mat');numFrames3 = size(vidFrames3_2,4);
%creates a spatial filter, initializing

filter = zeros(480,640);
filter(184:338,264:510) = 1;
mean_x3 = zeros(1,length(numFrames3));
mean_y3 = zeros(1,length(numFrames3));

for j = 1:numFrames3

    X = vidFrames3_2(:,:,:,j);
    %imshow(X); drawnow

    filter_uint8 = uint8(filter); %converts filter to an uint8 type
    gray_vid3 = rgb2gray(X); %turns to grayscale
    filt_vid3 = gray_vid3.*filter_uint8; %applies the spatial filter

    %thresh = filt_vid1 > 250; %0.97
    %could also binarize it and drop the tolerance
    thresh = imbinarize(filt_vid3,0.93);
    %imshow(thresh)

    %finds all non-zero vectors
    indeces = find(thresh);

    %finds the matrix/vectors
    [Y, X] = ind2sub(size(thresh),indeces);

    %finds the centroid!
    mean_x3(j) = mean(X);
    mean_y3(j) = mean(Y);

end
image_y3 = 480-mean_y3;
%invert the mean_y1 because the axis of a picture is upside down
image_x3 = 640 - mean_x3;




plot(1:numFrames1,mean_x1); axis([0 360 0 400]);hold on;
plot(1:numFrames2,mean_x2); hold on; plot(1:numFrames3,400-mean_y3);
legend('1.2','2.2','3.2')
title('Case 2: Unsynced Original Data');
```

```
xlabel('Frame Number');
ylabel('XY Displacement (pixels)');



%%
plot(1:numFrames1,image_y); axis([0 360 0 400]); hold on;
plot(1:numFrames2,image_y2); hold on; plot(1:numFrames3,image_x3);
legend('1_1','2_1','3_1')

min1 = 13; min2 = 37; min3 = 16; width = 301;
% 301
% 319
% 311
shift_mean1 = mean_y1(min1:min1+width);
shift_mean2 = mean_y2(min2:min2+width);
shift_mean3 = mean_x3(min3:min3+width);

set(0, 'DefaultLineLineWidth', 2);

figure();
subplot(2,2,1);
plot(1:302,480 - shift_mean1); hold on;
plot(1:302,480 - shift_mean2); hold on;
plot(1:302,640 - shift_mean3); hold on;
legend('1.2','2.2','3.2')
title('Case 2: Synced Original Data');
xlabel('Frame Number');
ylabel('Z Displacement (pixels)');

concat = [mean_x1(min1:min1+width);shift_mean1;mean_x2(min2:min2+width);...
    shift_mean2;mean_y3(min3:min3+width);shift_mean3];
%mean weights something less, normalize, average out teh center prosition


test1mat = [concat(1,:)-mean(concat(1,:));concat(2,:)-mean(concat(2,:));...
    concat(3,:)-mean(concat(3,:));concat(4,:)-mean(concat(4,:));...
    concat(5,:)-mean(concat(5,:));concat(6,:)-mean(concat(6,:))];


[u,s,v] = svd(test1mat/sqrt(301));

lambda = diag(s).^2; % produce diagonal variances
%Y= test1mat;
Y = u'*test1mat;
%Y = u'*test1mat;
sig= diag(s);
subplot(2,2,[2 4]);
plot (1:6 ,lambda/sum(lambda), 'mo ');
title (" Case 2: Energy Distribution");
xlabel ("Singular Values"); ylabel ("Energy");

subplot(2,2,3);
%plot(1:381,Y(1,:),1:381,Y(2,:),1:381,Y(3,:),1:381,Y(4,:),1:381,Y(5,:),1:381,Y(6,:));
plot(1:302,Y(1,:),1:302,Y(2,:),1:302,Y(3,:))
%,1:302,Y(4,:),1:302,Y(5,:),1:302,Y(6,:));
legend('PCA1','PCA2','PCA3');
title('Case 2: Principal Component Directions');
xlabel('Frame Number');
ylabel('Displacement (pixels)');
```