Principal Component Analysis

Yuanheng (Jonathan) Zhao

February 17th, 2020

Abstract

In this report, we introduce the backgrounds of principal component analysis (PCA), and practice it on four cases of a spring-mass system. We will examine the diagonal variance content of each of the principal components and the projections on their basis, and compare them with the original oscillation behavior of the system.

Sec I: Introduction and Overview

Linear algebra, such as SVD we will discuss, plays an important role in many applications of mathematics, physics, and engineering fields, as well as in data analysis and computation. We focus on a spring-mass example to see how we can apply some concepts to make analysis.

In this example, a bucket is suspended with a spring and oscillates. There is a light attached at the top of the bucket, which helps us to track the bucket. Three video cameras collect data about its motion in order to ascertain its governing equations. We are given datasets produced by the three cameras in four cases, each with somewhat oversampled. In the following sections, we are going to explore the PCA method on our spring-mass example and the effects of noise on the algorithm.

Sec II: Theoretical Background

2.1 Eigenvalues & Eigenvectors

Assume A is an $n \times n$ matrix, v is an $n \times 1$ vector, and λ is a scalar. And when we get a following form, we say that λ is the eigenvalue, and v is the eigenvector of matrix A (or transformation).

$$A\mathbf{v} = \lambda\mathbf{v} \tag{2.1.1}$$

By definition, an eigenvector of a linear transformation is a nonzero vector that changes at most by a scalar factor, eigenvalue λ, when that linear transformation is applied to it.

2.2 Singular Value Decomposition

A singular value decomposition (SVD) is a factorization of a matrix into several specific components, as the following form:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{2.2.1}$$

with the three components:

$$\begin{aligned} &\mathbf{U} \in \mathbb{C}^{m\times m} \text{ is unitary} \\ &\mathbf{V} \in \mathbb{C}^{n\times n} \text{ is unitary} \\ &\mathbf{\Sigma} \in \mathbb{R}^{m\times n} \text{ is diagonal} \end{aligned}$$

The diagonal entries of Σ are nonnegative and ordered from largest to smallest so that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$, and every matrix A has a singular value decomposition with singular values σ_j uniquely determined. The SVD of the matrix A thus shows that the matrix first applies a unitary transformation preserving the unit sphere via V^*. This is followed by a stretching operation that creates an ellipse with principal semiaxes given by the matrix Σ. Finally, the generated hyperellipse is rotated by the unitary transformation U.

2.3 Covariance Matrix

Covariance of two variables X and Y indicate the relationships, or correlation, between the two, which is defined as the following:

$$\operatorname{cov}(X,Y) = \frac{\sum_{i=1}^{n}\left(X_i - \bar{X}\right)\left(Y_i - \bar{Y}\right)}{n-1} \tag{2.3.1}$$

In statistics, strongly statistically dependent variables can be considered as redundant observations of the system. Covariance matrix can be helpful when we try to identify redundant data with higher dimensions. To calculate covariance matrix, we use the following formula:

$$\mathbf{C_X} = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T \tag{2.3.2}$$

The diagonal terms represent the variances of particular measurements and the off-diagonal terms are the covariances between measurement types. Note that the large diagonal terms typically represent the dynamics of interest, since the large variance suggests strong fluctuations in that variable, while small diagonal terms represent non-interesting parts.

Working with transformed variable Y, where $Y = U * X$ or $X' * V$, we consider its covariance as:

$$C_Y = \frac{1}{n-1}\Sigma^2 \tag{2.3.3}$$

2.4 Principal Component Analysis

Principal Component Analysis (PCA) is one of the major applications of SVD, which produces low-dimensional reductions of the dynamics and behavior when the governing equations are not known.

The following key steps are involved in applying the method to experimental data sets:

1) Organize the data into a matrix $A \in C^{m*n}$ where m is the number of measurement types (or probes) and n is the number of measurements (or trials) taken from each probe.
2) Subtract off the mean for each measurement type or row.
3) Compute the SVD and singular values of the covariance matrix to determine the principal components.

In our spring-mass system, we have three cameras recording the motion of a mass, each producing a 2-D representation of data. One thing we will consider is redundancy. And the other key measure is SNR, the signal-to-noise ratio, where the ratio is given as the ratio of variances of the signal and noise fields. A higher SNR suggests almost noiseless data while a low SNR suggests corrupted data by noise.

Sec III: Algorithm Implementation & Development

The procedure to analyze each case, with videos taken by three cameras, is generally similar, so we will take a look at a single procedure of processing any of the cases (tests).

Part 1: Load the data

First of all, we load the data for a single test, e.g. 'cam*i*_1.mat' with *i* = 1, 2, and 3. What we get are three 4-D datasets representing data points of rgb values over time. In order to focus on the motion of the bucket and reduce the resources consumed by unrelated data, we design a filter for each set of frames to filter out the areas that the bucket never enters. The filter is set to be a 480×640 matrix, as the same size of a video frame, and with elements within a specified square of area to be 1 and elsewhere to be 0. The area to be cropped is defined by observing each video many times.

Using the *size* command, we get the number of frames of a video (each one has a different number of frames). For each frame, we convert the rgb values representing that frame into grayscale by using *rgb2gray* and *double* command. Then we multiply the data in grayscale with the filter to get the cropped data.

Since there is a light attached on the top of the bucket, we can track the bucket by looking for the data points with maximum values in grayscale matrix over different points of time (higher grayscale values indicate lighter). We get these points by filtering out those with grayscale values less than a scale we specify (240 at most of time and 230 for the last one). Then we find the indices of these points via command *find*. After averaging the points, we get and save all the coordinates representing the center of the bucket at each frame, or points of time.

The steps above are written into a function called *load_cropped_data*, and we will apply it to each video.

Part 2: Synchronize the three cameras

The videos we just process have different frame numbers and are not in sync, thus we adjust the first frame in each of the three videos to be the frame with the lowest y coordinate. Then we identify the shortest video (video 1 for the first three tests and video 3 for the last one), and trim the other two data matrices to make the three data matrices the same frame number. We collect the three data matrices to form a single matrix, which has 6 rows incorporating the data from the three cameras over time.

Part 3: SVD & Results

In order to center the data matrix, we subtract the mean of each row from the data, take the transpose of it and divide by $\sqrt{n-1}$. Then we take SVD of it to get the components U, S, and V. We get the diagonal variance via the command *lambda=diag(s).^2* and produce the principal components projection via the command *Y = collected_data' * V*. Then we can plot the diagonal variances to see the importance of each principal component, and we can also plot the projections to reconstruct our observations.

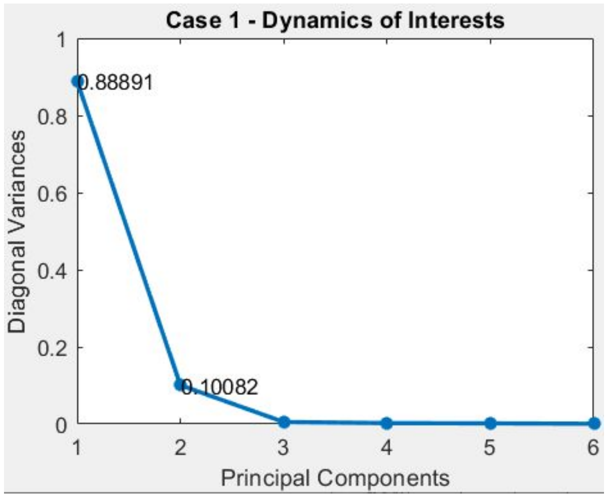
Sec IV: Computational Results

Test 1 (Ideal Case)

From Figure 1, we can see that we only have the first principal component which holds the most of the diagonal variances, or energy/dynamics of interests (about 0.889), and other components have very low dynamics. The projection onto the first principal component basis forms a similar result as the original data measured, indicating a simple harmonic motion. These fit the condition of case 1, and show that the system can be reproduced by the first principal component.

Test 2 (Noisy Case)

Based on Figure 3, we might consider two principal components significant, which correspond to values of 0.643 and 0.170. (We might also consider three principal components.) Since there exists noise in our system, the original data and the projection to the principal basis show noise as the bucket oscillates. We can see that PCA does not eliminate noise, but it can reduce noise, indicating an oscillatory behavior.



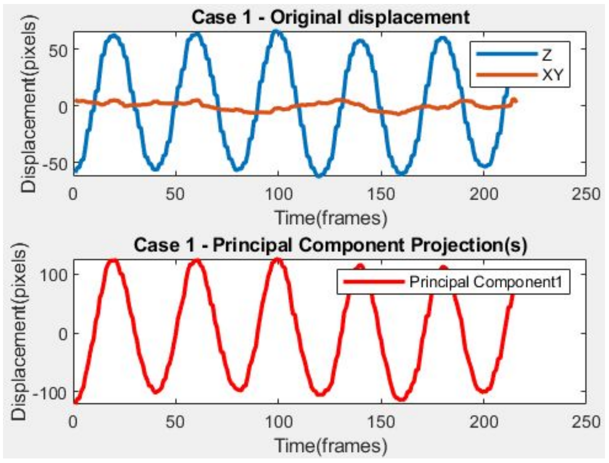
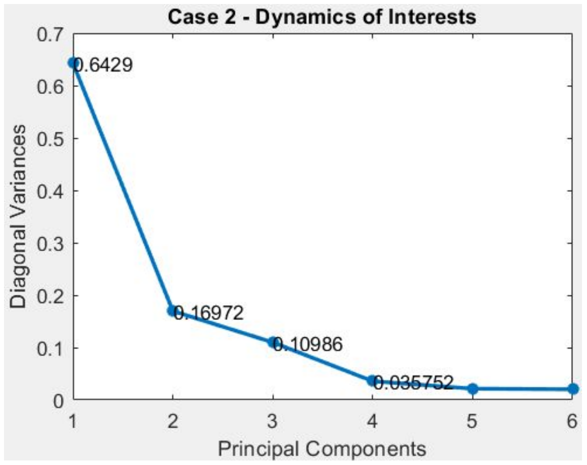


Figure 1: Plot of diagonal var of PCs in test 1

Figure 2: Plot of original displacement & in new basis, test 1



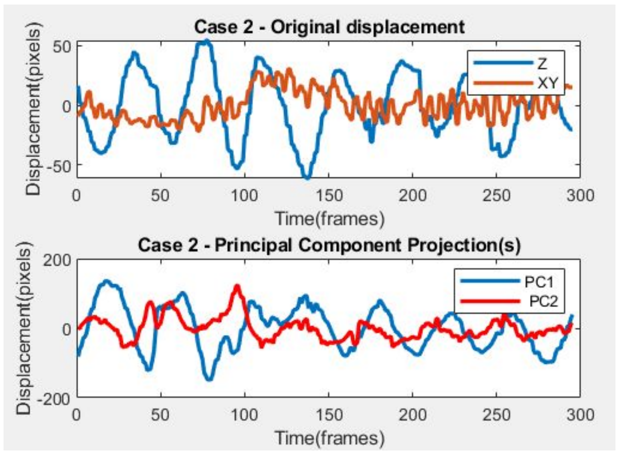


Figure 3: Plot of diagonal var of PCs in test 2

Figure 4: Plot of original displacement & in new basis, test 2

Test 3 (Horizontal displacement)

From Figure 5, the first four principal components are considered significant, which have values of 0.500, 0.227, 0.163, and 0.104. The first component holds about half of the energy and the other three hold relatively important amounts of variances. The projections from Figure 6 indicates there exists simple harmonic motion, pendulum motion, and noise caused by the camera shaking.

Test 4 (Horizontal displacement and rotation)

From Figure 7, we can see there exist three principal components hold relatively higher variances, which correspond to the values of 0.691, 0.180, and 0.090. The projections onto principal component basis have relatively smooth curves and fit the case, as the system experiences oscillatory motion, rotation, and pendulum motion in x-y plane. It shows that PCA has achieved the explanation of the multi-dimensional identities of the spring-mass system.

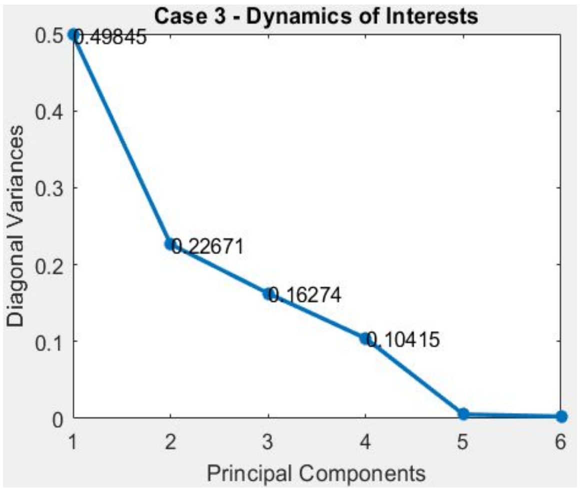
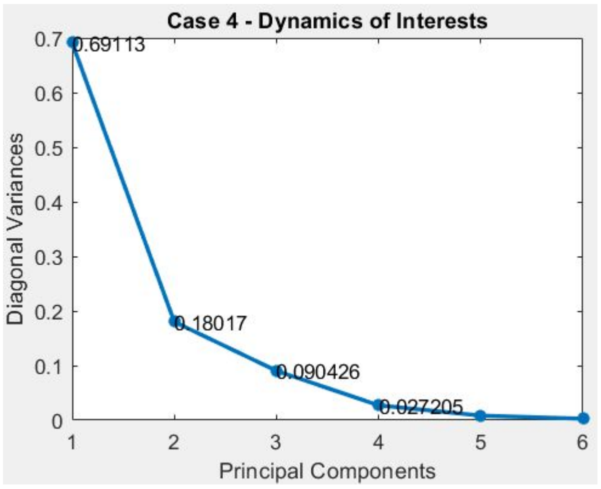


Figure 5: Plot of diagonal var of PCs in test 3

Figure 6: Plot of original displacement & in new basis, test 3



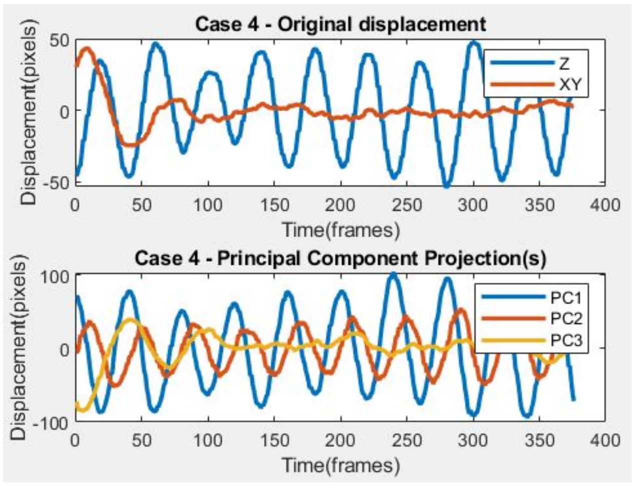


Figure 7: Plot of diagonal var of PCs in test 4

Figure 8: Plot of original displacement & in new basis, test 4

Appendix A: MATLAB functions used

[M, I] = min: returns the linear index into A that corresponds to the minimum value in A.
find: returns a vector containing the linear indices of each nonzero element in array X.
size: returns a row vector whose elements are the lengths of the corresponding dimensions of A.
repmat: returns an array containing n copies of A in the row and column dimensions.
diag: returns a square diagonal matrix with the elements of vector v on the main diagonal.
[U,S,V] = svd: performs a singular value decomposition of matrix A, such that A = U*S*V'.

Appendix B: MATLAB codes

```
%% Load the data (videos) for Test 1/Case 1
clear; close all; clc

load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')
% implay(vidFrames1_1)
%%
filter = zeros(480 ,640);
filter(200:430, 300:400) = 1;  % Filter for Case1 - Camera1
data1 = load_cropped_data(vidFrames1_1, filter, 240);
%%
filter = zeros(480 ,640);
filter(100:390, 240:350) = 1;  % Filter for Case1 - Camera2
data2 = load_cropped_data(vidFrames2_1, filter, 240);
%%
filter = zeros(480 ,640);
filter(230:330, 260:480) = 1;  % Filter for Case1 - Camera3
data3 = load_cropped_data(vidFrames3_1, filter, 240);
%%
collected_data = collect(data1, data2, data3);
%% Results for Case 1
[m,n]= size(collected_data);  % compute data size

collected_data = collected_data - repmat(mean(collected_data, 2),1,n);  % subtract
mean
[U,S,V]= svd(collected_data'/sqrt(n-1));  % perform the SVD
lambda = diag(S).^2;  % produce diagonal variances
Y = collected_data' * V;  % produce the principal components projection

figure(1)
plot(1:6, lambda/sum(lambda), '-*', 'Linewidth', 2);
title("Case 1 - Dynamics of Interests");
xlabel("Principal Components"); ylabel("Diagonal Variances");
y = lambda/sum(lambda);
for i=1:2
   text(i,y(i),num2str(y(i)));
end

figure(2)
subplot(2,1,1)
```

```
plot(1:216, collected_data(2,:), 1:216, collected_data(1,:), 'Linewidth', 2)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 1 - Original displacement");
legend("Z", "XY")
subplot(2,1,2)
plot(1:216, Y(:,1),'r','Linewidth', 2)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 1 - Principal Component Projection(s)");
legend("Principal Component1")

%% Load the data for Test 2/Case 2
clear all; close all; clc;

load('cam1_2.mat')
load('cam2_2.mat')
load('cam3_2.mat')

%%
filter = zeros(480 ,640);
filter(200:400, 300:430) = 1;  % Filter for Case2 - Camera1
data1 = load_cropped_data(vidFrames1_2, filter, 240);
%%
filter = zeros(480 ,640);
filter(50:420, 180:440) = 1;  % Filter for Case2 - Camera2
data2 = load_cropped_data(vidFrames2_2, filter, 240);
%%
filter = zeros(480 ,640);
filter(180:330, 280:470) = 1;  % Filter for Case2 - Camera3
data3 = load_cropped_data(vidFrames3_2, filter, 240);
%%
collected_data = collect(data1, data2, data3);
%% Results for Test 2
[m,n]= size(collected_data);  % compute data size

collected_data = collected_data - repmat(mean(collected_data, 2),1,n);  % subtract
mean
[U,S,V]= svd(collected_data'/sqrt(n-1));  % perform the SVD
lambda = diag(S).^2;  % produce diagonal variances
Y = collected_data' * V;  % produce the principal components projection

figure(3)
plot(1:6, lambda/sum(lambda), '-*', 'Linewidth', 2);
title("Case 2 - Dynamics of Interests") ;
xlabel("Principal Components"); ylabel("Diagonal Variances");
y = lambda/sum(lambda);
```

```
for i=1:4
   text(i,y(i),num2str(y(i)));
end

figure(4)
subplot(2,1,1)
plot(1:295, collected_data(2 ,:), 1:295, collected_data(1 ,:) ,'Linewidth', 2)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
legend("Z", "XY")
title("Case 2 - Original displacement");
subplot(2,1,2)
plot(1:295, Y(:,1), 1:295, Y(:,2), 1:295, Y(:,3), 'r', 'Linewidth', 2)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 2 - Principal Component Projection(s)");
legend("PC1" , " PC2")

%% Load the data for Test 3/Case 3
clear all; close all; clc;

load('cam1_3.mat')
load('cam2_3.mat')
load('cam3_3.mat')
%%
filter = zeros(480 ,640);
filter(240:420, 280:380) = 1;  % Filter for Case3 - Camera1
data1 = load_cropped_data(vidFrames1_3, filter, 240);
%%
filter = zeros(480 ,640);
filter(180:380, 240:400) = 1;  % Filter for Case3 - Camera2
data2 = load_cropped_data(vidFrames2_3, filter, 240);
%%
filter = zeros(480 ,640);
filter(180:330, 240:480) = 1;  % Filter for Case3 - Camera3
data3 = load_cropped_data(vidFrames3_3, filter, 240);
%%
collected_data = collect(data1, data2, data3);
%% Results for Test 3
[m,n]= size(collected_data);  % compute data size

collected_data = collected_data - repmat(mean(collected_data, 2),1,n);  % subtract
mean
[U,S,V]= svd(collected_data'/sqrt(n-1));  % perform the SVD
lambda = diag(S).^2;  % produce diagonal variances
Y = collected_data' * V;  % produce the principal components projection
```

```
figure(5)
plot(1:6, lambda/sum(lambda), '-*', 'Linewidth', 2);
title("Case 3 - Dynamics of Interests");
xlabel("Principal Components"); ylabel("Diagonal Variances");
y = lambda/sum(lambda);
for i=1:4
   text(i,y(i),num2str(y(i)));
end

figure(6)
subplot(2,1,1)
plot(1:235, collected_data(2 ,:), 1:235, collected_data(1,:),'Linewidth', 2)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 3 - Original displacement");
legend("Z", "XY")
subplot(2,1,2)
plot(1:235,Y(:,1),1:235,Y(:,2),1:235,Y(:,3),1:235,Y(:,4),'r','Linewidth',2)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 3 - Principal Component Projection(s)");
legend("PC1", "PC2", "PC3", "PC4")

%% Load the data for Test 4/Case 4
clear all; close all; clc;

load('cam1_4.mat')
load('cam2_4.mat')
load('cam3_4.mat')
%%
filter = zeros(480 ,640);
filter(230:440, 330:460) = 1;  % Filter for Case4 - Camera1
data1 = load_cropped_data(vidFrames1_4, filter, 240);
%%
filter = zeros(480 ,640);
filter(100:360, 230:410) = 1;  % Filter for Case4 - Camera2
data2 = load_cropped_data(vidFrames2_4, filter, 240);
%%
filter = zeros(480 ,640);
filter(140:280, 320:500) = 1;  % Filter for Case4 - Camera3
data3 = load_cropped_data(vidFrames3_4, filter, 230);
%%
[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,:);

% Trim the data to make them a consistent length.
% For Test4, the video recorded by camera 3 is the shortest.
% Thus trim the other two as the length of video 3.
data1 = data1(1:length(data3), :);
data2 = data2(1:length(data3), :);

collected_data = [data1'; data2'; data3'];

%% Results for Test 4
[m,n]= size(collected_data);  % compute data size

collected_data = collected_data - repmat(mean(collected_data, 2),1,n);  % subtract
mean
[U,S,V]= svd(collected_data'/sqrt(n-1));  % perform the SVD
lambda = diag(S).^2;  % produce diagonal variances
Y = collected_data' * V;  % produce the principal components projection

figure(7)
plot(1:6, lambda/sum(lambda), '-*', 'Linewidth', 2);
title("Case 4 - Dynamics of Interests");
xlabel("Principal Components"); ylabel("Diagonal Variances");
y = lambda/sum(lambda);
for i=1:4
   text(i,y(i),num2str(y(i)));
end

figure(8)
subplot(2,1,1)
plot(1:376 , collected_data(2,:), 1:376, collected_data(1,:),'Linewidth', 2)
ylabel("Displacement(pixels)") ; xlabel("Time(frames)") ;
title("Case 4 - Original displacement") ;
legend("Z", "XY")
subplot(2,1,2)
plot(1:376, Y(:,1), 1:376, Y(:,2), 1:376, Y(:,3), 'Linewidth', 2)
ylabel("Displacement(pixels)"); xlabel("Time(frames)") ;
title("Case 4 - Principal Component Projection(s)") ;
legend("PC1", "PC2", "PC3")



% Collect and arrange the data from three cameras into a single matrix
% Thus the matrix incorporates data points of all measurement types
% taken by each camera over time.
```

```
function collected_data = collect(data1, data2, data3)
   [~,I] = min(data1(1:20,2));
   data1 = data1(I:end,:);
   [~,I] = min(data2(1:20,2));
   data2 = data2(I:end,:);
   [~,I] = min(data3(1:20,2));
   data3 = data3(I:end,:);

   % Trim the data to make them a consistent length.
   % For Test1, 2, and 3, the video recorded by camera 1 is (always) the shortest.
   % Thus trim the other two as the length of video 1.
%    disp(length(data1));
%    disp(length(data2));
%    disp(length(data3));
   data2 = data2(1:length(data1), :);
   data3 = data3(1:length(data1), :);

   collected_data = [data1'; data2'; data3'];
end



% Load and crop the videos based on the filter specified
function data = load_cropped_data(vidFrames, filter, scale)
   numFrames = size(vidFrames,4);
   data = zeros(numFrames, 2);
   for j = 1:numFrames
      X = vidFrames(:,:,:,j);
      Xg = double(rgb2gray(X));
      X_cropped = Xg .* filter;
      threshold = X_cropped > scale; % light

      [Y, X] = find(threshold);
      data(j,1) = mean(X);
      data(j,2) = mean(Y);
%       subplot(1 ,2 ,1)
%       imshow(uint8((threshold * 255))); drawnow
%       subplot(1 ,2 ,2)
%       imshow(uint8(X_cropped)); drawnow
   end
end
```