Principle Component Analysis
Physical Model Extraction from Oversampled and Noisy Systems

Dino de Raad

April 11, 2019

Abstract

The Singular Value Decomposition and Principle Component Analysis are described. The Principal Component Analysis is then used to extract the dynamics of a simple physical system, a spring-mass system with pendulum motion, from a convoluted and redundant set-up.

I Introduction and Overview

Matrix decomposition techniques are used in a wide variety of problems. For example, the LU decomposition simplifies computational Gaussian elimination in the worst cases. The Singular Value Decomposition (SVD) allows for any matrix to be decomposed. Thus we can use it for rectangular image analysis. Specifically, we consider the Principle Component Analysis, via which we reduce the total modes needed to describe data in a stream. Specifically, we consider a redundant system with three cameras recording a spring-mass and pendulum system and attempt to reduce it to the familiar physical dynamics.

II Theoretical Background

The Singular Value Decomposition (SVD) intuitively captures the action of matrix multiplication. Matrix multiplication can be thought of as a rotation and a scaling of a vector, which are both simple matrix operations. Rotation can be achieved by the rotation matrix

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

in 2D. and scaling by

$$\mathbf{B} = \begin{bmatrix} \alpha & 0 \\ 0 & \alpha \end{bmatrix},$$

where θ is the angle of rotation and α is the scaling factor. However, not all matrices are square. If we are generalize this idea for all linear transforms (this is what matrix multiplication is) we need to capture projection into higher and lower dimensionality.

We define the matrix $\mathbf{A}^{m\times n}$ as having rank r. Then the singular value decomposition is

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

where $\mathbf{V}^{*n\times r}$ projects a vector into r dimensional space, $\mathbf{\Sigma}^{r\times r}$ is the diagonal matrix containing the r singular values ordered by magnitude, and $\mathbf{U}^{m\times r}$ projects into the m dimensional output space. We can think of them (in order) as a rotation, a scaling, and a rotation, to satisfy our intuition. One key property worth mentioning is that a truncation of the singular values provides the best approximation in an L2 sense given that many singular values of any matrix decomposition.

The Principle Component Analysis (PCA) exploits the idea of the rank-r truncation of the SVD. We use it to look for low-dimensional representations of the data, and in an L2 sense, the best low-dimensional representations are exactly those of the SVD. If we have m measurements streams and n samples from each stream, the $m \times n$ matrix $\mathbf{X}$ is of great importance; if we can diagonalize this matrix and extract the proper basis functions, we can begin to characterize a physical system via truncation of noisy modes. We consider $\mathbf{Y} = \mathbf{U}^*\mathbf{X}$, the projection of $\mathbf{X}$ onto the principal components. The simplified co-variance matrix of $\mathbf{Y}$ is

$$\mathbf{C}_\mathbf{Y} = \frac{1}{n-1}\mathbf{\Sigma}^2,$$

so the SVD gives a simple way to diagonalize $\mathbf{X}$. Both this co-variance matrix and the projection onto the principle components are useful for visualizing and analyzing the data.

III Algorithm Implementation and Development

Algorithm 1

The purpose of this algorithm is to determine the (x, y) coordinates of the paint can in the videos at every frame.

For each video:

1. Load video and determine length in frames.
2. For each frame up to the video's length:
    - Convert frame to black and white.
    - Since the paint can is brighter than most elements of any particular image, and there is a bright light atop the can, filtering all brightness values below some threshold is sufficient to generally locate the can. Determine some appropriate threshold (this will vary in between videos as lighting conditions and background objects appear and disappear). All values satisfying this condition can be made 1 and all less 0.
    - Remove frames of static disturbances (0 them out); these are not the signal of interest.
    - Compute the centroid (Algorithm 2) of these bright pixels.
    - Record the centroids.
3. Clear the video from memory.

Algorithm 2

This algorithm computes the centroid (x_c, y_c) of a matrix $\mathbf{A}$. We assume the matrix is an $x - y$ plane with a weight at each grid-point.

1. The size of the axial vector x is determined to the size of the first dimension of the matrix A and y is the axial vector corresponding to the size of the second dimension.
2. Find the `meshgrid` of x and y, which is the X, Y grid associated with $\mathbf{A}$ where each element of $\mathbf{X}$ corresponds to an x coordinate of $\mathbf{A}$, and likewise with $\mathbf{Y}$.
3. Find the mean value of $\mathbf{A}$, $\bar{A}$.
4. The x-centroid and y-centroid are computed similarly:

$$x_c = \frac{\text{mean}_{ij}(a_{ij} * x_{ij})}{\bar{A}} \qquad y_c = \frac{\text{mean}_{ij}(a_{ij} * y_{ij})}{\bar{A}}$$

   where the a_{ij}, x_{ij} and y_{ij} are the entries of $\mathbf{A}$, $\mathbf{X}$ and $\mathbf{Y}$ at the ith row and jth column.

Algorithm 3

If the videos do not begin at the same time, we can use this algorithm with some assumptions to synchronize the videos.

1. Determine a common feature of each of the three coordinate sets. We will consider in this case regions of maximal displacement.
2. Determine the local maxima/minima at these points, and use these to generate a frame number where these occur in each of the videos.
3. Select the maximum of these extreme event locations and subtract it from all of the extreme event locations. These will be the lengths l_k for $k = a, b, c$ corresponding to videos a, b, and c.
4. Remove all of the frames occuring before the respective event locations by removing the first l_k frames from x_k, y_k, and t_k.
5. Trim from the end of all the time series so that they are the correct length.

Algorithm 4

This algorithm extracts the principle components and singular values (variances) of the Matrix $\mathbf{X}$, a special matrix whose rows are the x and y positions of the objects of interest, so that the columns are time slices at which those positions occur.

1. Take the singular value decomposition (Matlab `svd`) of the demeaned data $\mathbf{X}^T$. This gives the matrices $\mathbf{U}$, $\mathbf{S}$, and $\mathbf{V}$.
2. Project the data onto the principle component directions with $Y = k * (\mathbf{U}^T * X)$ where $k = \frac{1}{\sqrt{n-1}}$.
3. The variances are given by the diagonal elements $\mathbf{S}$ squared scaled by k as given in 2.

IV Computational Results

In general, we are able to extract the primary oscillatory mode, reducing the massively redundant system to the simple 1D harmonic motion. As we see in the first sub-figure of Figures 1, 2, 4, and 5, the primary motion of the spring-mass system is well described by a sinusoidal curve, and this class of curves solves the fundamental differential equations associated with the system. Looking at the tests individually:

1. **Ideal Case (Figure 1):** The singular value magnitude suggests that most of the energy of the system is expressed in the first mode. Indeed, the jump from the first principle component to the second is of two orders of magnitude. This is well reflected in the principle component projections, where the first mode shows the spring-mass behaviour, but all subsequent modes don't show any periodic behavior.
2. **Noisy Case (Figure 2):** As expected when noise is introduced, the drop in magnitude of the singular values is reduced significantly. Periodic behaviour is clear in the first mode and somewhat evident in the second mode of the projections. To remove some noise from the projection and find the underlying shape of the motion, filtering of the signal was performed in the Fourier domain accounting for the structure of the signal (Figure 3).
3. **Horizontal Displacement Case (Figure 4):** Somewhat unexpectedly, 3 significant principle components are extracted and clear signals are evident in all of them, despite the fact that theoretically the motion of the can is characterized by the spring-mass and pendulum motions only. The periodicity of the second two principle component projections is in line with the pendulum motion, but is still confusing to interpret without suggesting there is a minimal amount of rotation.

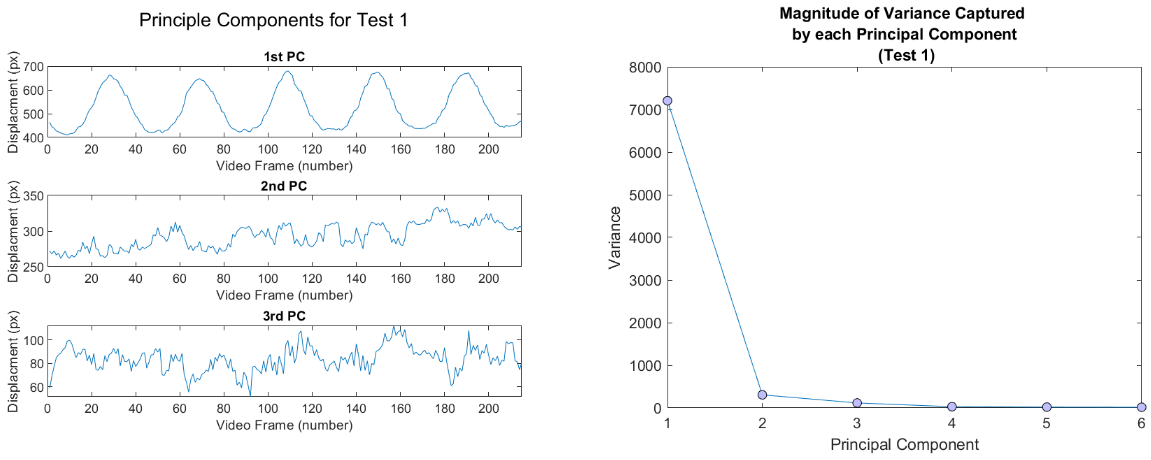


Figure 1: The projections of the first 3 principle components for test 1, and their respective variances. The first principle component is nearly 10^2 times bigger than the second, suggesting that the major dynamics of the system are captured in the first projection. Indeed, the spring-mass oscillations are evident only the first mode, and no clear meaning can be ascribed to the latter projections, as they reflect dynamics of the noise in the signal.

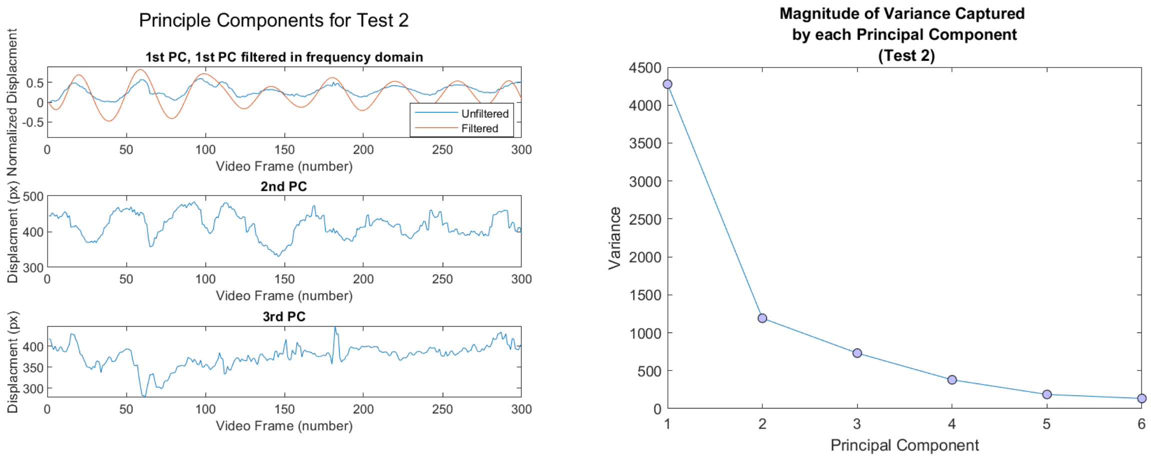


Figure 2: The projections of the first 3 principle components for test 2, and their respective variances. Special treatment has been given to this case, since the noise is so overwhelmingly evident even in the first principle projection. In orange, the signal has been filtered (explained in Figure 3) and reveals the familiar spring-mass motion. Some components of periodic motion are evident in the second projection, but are not particularly regular. Most likely the second, third, and fourth components characterize elements of the camera motion of the 3 cameras.

4. **Horizontal Displacement and Rotation Case (Figure 5):** The first two singular components describe the spring mass motion and pendulum motion respectively. The third and fourth characterize undulations in the $x - y$ plane. As we can see, the PCA fails to capture the rotation exceedingly well with the current data structure. One of the main problems is that the algorithm used to track the can does not choose a consistent point on the can, but rather tries to track the center of the object. As a result, rotation of the can itself is not well-tracked. Additionally, the camera placements may lend themselves poorly to characterizing the $x - y$ plane, leading to the noise in the reconstruction along with the interference of the tracking algorithm.

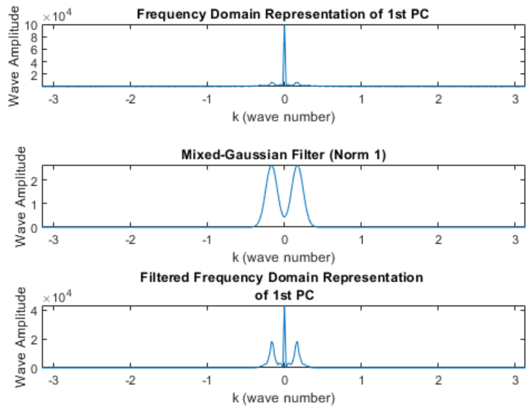


Figure 3: The Fourier decomposition (absolute value) of the 1st PC in test 2. The filter is the summation of 2 Gaussians centered at the first non-zero dominant modes. The filter widths were chosen such that the zero-mode is twice the height of the other dominant modes.

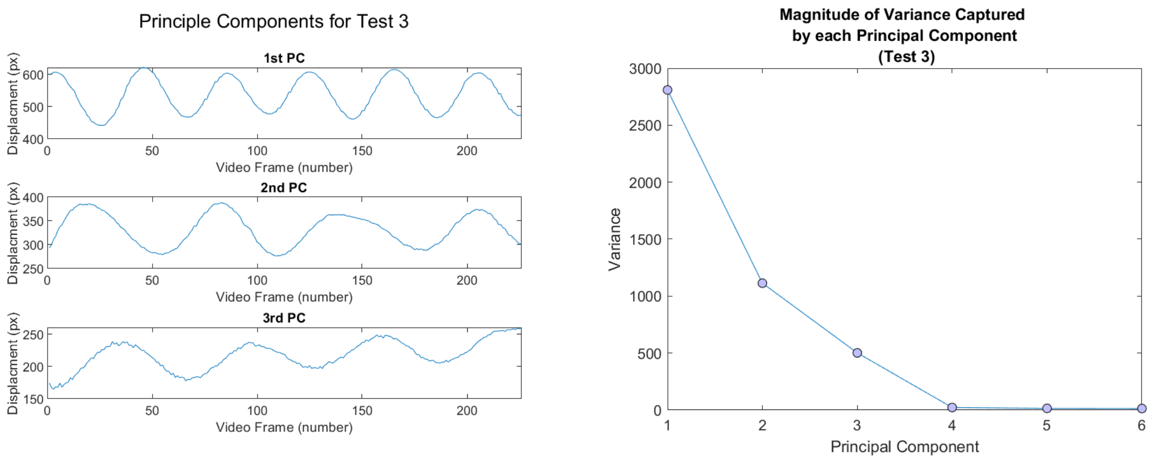


Figure 4: The projections of the first 3 principle components for test 3, and their respective variances. The second and third components are significant, and show clear periodic behaviour associated with the pendulum motion, since their periodicity is synchronized but altogether different from the first.

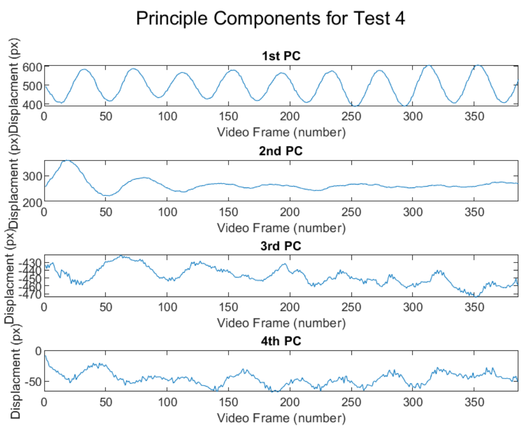
V Summary and Conclusions

The Principle Component Analysis extracts exceedingly well the fundamental motions in the videos in much fewer dimensions than are initially present. Noise clearly affects the result, and it becomes evident that some scenarios do not allow for good principle component analysis. With just the slight camera shake introduced in the second set of videos, the principle components (all of them) become noticeably noisier. Overall the method is an effective way to extract the dynamics of a redundant sensor array.

Appendix A MATLAB functions

Function: Centroid

This function finds the centroid of a matrix `A` with weights at each data-point.



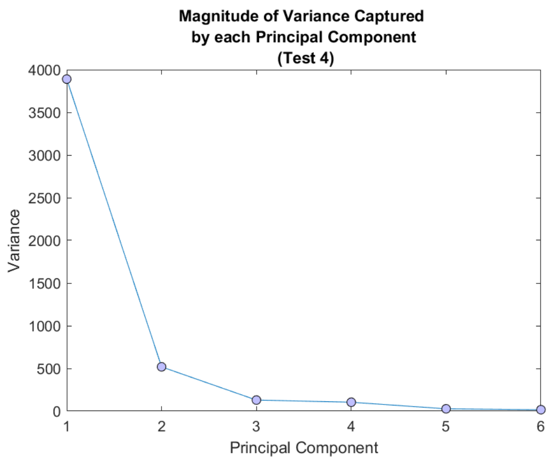


Figure 5: The projections of the first 4 principle components for test 4, and their respective variances. The second component characterizes the pendulum motion as it decays, and the third and fourth characterize motion in the $x - y$ plane, since they are synchronous but aperiodic with the first principle component. These signals are quite noisy, probably due to the position of the cameras poorly capturing complex motions in the $x - y$ space.

```
function [comX,comY] = centroid(A)
x = 1:size(A,2);
y = 1:size(A,1);
[X, Y] = meshgrid(x,y);
meanA = mean(A(:));
comX = mean(A(:).*X(:))/meanA;
comY = mean(A(:).*Y(:))/meanA;
end
```

Function: Gaussian Filter

This function generates a Gaussian with center `tau`, axis `t`, and standard deviation `a`.

```
function filter = Gaussianfilter(t,a,tau)

filter = 1/(sqrt(2*pi)*a)*exp(1/(2*a^2)*-(t-tau).^2);

end
```

Appendix B MATLAB codes

```
%% (test 1) Ideal Case
clear variables; close all; clc;
%% Cam 1_1


load cam1_1.mat

datasize = size(vidFrames1_1); % height, width, color channels, length

x_a = zeros([datasize(4) 1]); y_a = x_a;
```

```
for j=1:datasize(4)
    gryfrm = rgb2gray(vidFrames1_1(:,:,:,j));
    frm = gryfrm>=255;
    frm(1:200,1:300) = 0; % remove artifact
    %imshow(frm, []);
    %hold on
    [x_a(j), y_a(j)] = centroid(frm);
    %scatter(xcom, ycom, 50, 'x');
    %hold off
    %drawnow
end

clear vidFrames1_1;
'Cam1_1 done.'

%% Cam 2_1

load cam2_1.mat;

x_b = zeros([datasize(4) 1]); y_b = x_b;

for j=1:datasize(4)
    gryfrm = rgb2gray(vidFrames2_1(:,:,:,j));
    frm = gryfrm>=250;
    % imshow(frm, []);
    %hold on
    [x_b(j), y_b(j)] = centroid(frm);
    %scatter(xcom, ycom, 50, 'x');
    %hold off
    drawnow
end

clear vidFrames2_1;
'Cam2_1 done.'

%% Cam 3_1
load cam3_1.mat;

x_c = zeros([datasize(4) 1]); y_c = x_c;

for j=1:datasize(4)
    gryfrm = rgb2gray(vidFrames3_1(:,:,:,j));
    frm = gryfrm>=245;
    frm(:,1:200) = 0; % remove artifact
    %imshow(frm, []);
    %hold on
    [x_c(j), y_c(j)] = centroid(frm);
    %scatter(xcom, ycom, 50, 'x');
    %hold off
    drawnow
end

clear vidFrames3_1;
```

```
'Cam3_1 done.'

%% (test 1) Ideal Case Analysis

% Not all sensors started recording at the same time.
% We need to synchronize the recordings. This means we will have
% to synchronize the 1st shift in direction and trim frames not shared
% by all videos.


t_a = 1:length(x_a);
t_b = 1:length(x_b);
t_c = 1:length(x_c);

figure(3)
subplot(3,1,1)
[pksa, loca] = findpeaks(y_a, 'MinPeakDistance', 35);
findpeaks(y_a, 'MinPeakDistance', 35)
subplot(3,1,2)
[pksb, locb] = findpeaks(y_b, 'MinPeakDistance', 35); % 2nd peak
findpeaks(y_b, 'MinPeakDistance', 35)
subplot(3,1,3)
[pksc, locc] = findpeaks(x_c, 'MinPeakDistance', 35);
findpeaks(x_c, 'MinPeakDistance', 35)

firstpeaks = [loca(1) locb(2) locc(1)];

chopinit = firstpeaks - min(firstpeaks); %
achop = 1:chopinit(1);
x_a(achop) = []; y_a(achop) = []; t_a(achop) = [];
bchop = 1:chopinit(2);
x_b(bchop) = []; y_b(bchop) = []; t_b(bchop) = [];
cchop = 1:chopinit(3);
x_c(cchop) = []; y_c(cchop) = []; t_c(cchop) = [];

maxvidlength = min([length(t_a) length(t_b) length(t_c)]);
t = 1:maxvidlength;

x_a = x_a(t); x_b = x_b(t); x_c = x_c(t);
y_a = y_a(t); y_b = y_b(t); y_c = y_c(t);

figure(4)
subplot(3,1,1)
plot(t,y_a)
subplot(3,1,2)
plot(t,y_b)
subplot(3,1,3)
plot(t,x_c)

close all;

X = [x_a'
    y_a'
    x_b'
    y_b'
    x_c'
    y_c'];

% Code from the book:
[m,n]=size(X); % compute data size
[u,s,v]=svd((X-mean(X,2))/sqrt(n-1)); % perform the SVD
lambda=diag(s).^2; % produce diagonal variances
Y=u'*X; % produce the principal components projection


fig1 = figure(1);
sgtitle('Principle Components for Test 1')
subplot(3,1,1)
Y = Y';
plot(t, Y(:,1))
title('1st PC')
ylabel('Displacment (px)')
xlabel('Video Frame (number)')
V = axis;
V(1:2) = [0 length(t)];
axis(V)

subplot(3,1,2)
plot(t, Y(:,2))
V = axis;
V(1:2) = [0 length(t)];
axis(V)
ylabel('Displacment (px)')
xlabel('Video Frame (number)')
title('2nd PC')

subplot(3,1,3)
plot(t, Y(:,3))
V = axis;
V(1:2) = [0 length(t)];
axis(V)
ylabel('Displacment (px)')
xlabel('Video Frame (number)')
title('3rd PC')

fig2 = figure(2);
plot(1:length(lambda), lambda, '-o', 'MarkerEdgeColor','k',...
    'MarkerFaceColor',[0.75,0.75,1])
xticks(1:length(lambda))
ylabel('Variance')
xlabel('Principal Component')
title(sprintf('Magnitude of Variance Captured \n by each Principal Component \n (Test 1)'))

%% Saving

saveas(fig1,'PC1.png')
saveas(fig2,'V1.png')
```

```
%% (test 2) Noisy Case
clear variables; close all; clc;
%% Cam 1_2

load cam1_2.mat

datasize = size(vidFrames1_2); % height, width, color channels, length

x_a = zeros([datasize(4) 1]); y_a = x_a;

for j=1:datasize(4)
    gryfrm = rgb2gray(vidFrames1_2(:,:,:,j));
    frm = gryfrm>=240;
    frm(:,1:300) = 0; % remove artifact from left side
    frm(1:150,:) = 0; % remove artifact from top
    %imshow(frm, []);
    %hold on
    [x_a(j), y_a(j)] = centroid(frm);
    %scatter(xcom, ycom, 50, 'x');
    %hold off
    %drawnow
end

clear vidFrames1_2;
'Cam1_2 done.'

%plot(y_a)

%% Cam 2_2

load cam2_2.mat;

x_b = zeros([datasize(4) 1]); y_b = x_b;

for j=1:datasize(4)
    gryfrm = rgb2gray(vidFrames2_2(:,:,:,j));
    frm = gryfrm>=245;
    frm(1:175,1:175) = 0; % remove artifact from top left side
    frm(:,end-150:end) = 0; % remove artifact from right side
    %imshow(frm, []);
    %hold on
    [x_b(j), y_b(j)] = centroid(frm);
    %scatter(xcom, ycom, 50, 'x');
    %hold off
    drawnow
end

clear vidFrames2_2;
'Cam2_2 done.'

%plot(y_b)

%% Cam 3_2
load cam3_2.mat;
```

```
x_c = zeros([datasize(4) 1]); y_c = x_c;

for j=1:datasize(4)
    gryfrm = rgb2gray(vidFrames3_2(:,:,:,j));
    frm = gryfrm>=240;
    frm(:,1:250) = 0; % remove artifact from left side
    %imshow(frm, []);
    %hold on
    [x_c(j), y_c(j)] = centroid(frm);
    %scatter(xcom, ycom, 50, 'x');
    %hold off
    drawnow
end

clear vidFrames3_2;
'Cam3_2 done.'

%plot(x_c)

%% (test 1) Noisy Case Analysis

% Not all sensors started recording at the same time.
% We need to synchronize the recordings. This means we will have
% to synchronize the 1st shift in direction and trim frames not shared
% by all videos.

t_a = 1:length(x_a);
t_b = 1:length(x_b);
t_c = 1:length(x_c);

figure(3)
subplot(3,1,1)
[pksa, loca] = findpeaks(y_a, 'MinPeakDistance', 35);
findpeaks(y_a, 'MinPeakDistance', 35)
subplot(3,1,2)
[pksb, locb] = findpeaks(y_b, 'MinPeakDistance', 35); % 2nd peak
findpeaks(y_b, 'MinPeakDistance', 35)
subplot(3,1,3)
[pksc, locc] = findpeaks(x_c, 'MinPeakDistance', 35);
findpeaks(x_c, 'MinPeakDistance', 35)

syncpeaks = [loca(end-1) locb(end-1) locc(end-1)];

chopinit = syncpeaks - min(syncpeaks); %
achop = 1:chopinit(1);
x_a(achop) = []; y_a(achop) = []; t_a(achop) = [];
bchop = 1:chopinit(2);
x_b(bchop) = []; y_b(bchop) = []; t_b(bchop) = [];
cchop = 1:chopinit(3);
x_c(cchop) = []; y_c(cchop) = []; t_c(cchop) = [];

maxvidlength = min([length(t_a) length(t_b) length(t_c)]);
t = 1:maxvidlength;
```

```
x_a = x_a(t); x_b = x_b(t); x_c = x_c(t);
y_a = y_a(t); y_b = y_b(t); y_c = y_c(t);

figure(4)
subplot(3,1,1)
plot(t,y_a)
subplot(3,1,2)
plot(t,y_b)
subplot(3,1,3)
plot(t,x_c)

close all;

X = [x_a'
    y_a'
    x_b'
    y_b'
    x_c'
    y_c'];

% Code from the book:
[m,n]=size(X); % compute data size
[u,s,v]=svd((X-mean(X,2))/sqrt(n-1)); % perform the SVD
lambda=diag(s).^2; % produce diagonal variances
Y=u'*X; % produce the principal components projection

figure(1)
subplot(3,1,1)
Y = Y';
plot(t, (Y(:,1)-Y(1,1))/max(abs(Y(:,1))))
title('1st PC, 1st PC filtered in frequency domain')
ylabel('Normalized Displacment')
xlabel('Video Frame (number)')

Y1t = fftshift(fft(Y(:,1)));
n = length(t); % fourier modes
L = t(end) - t(1); % length of video
k=(2*pi/L)*[0:(n/2-1) -n/2:-1]; ks=fftshift(k);
figure(5)
subplot(3,1,1)
plot(ks, abs(Y1t))
xlabel('k (wave number)')
ylabel('Wave Amplitude')
title('Frequency Domain Representation of 1st PC')
axis tight
sigma = .075;
mu = .1681;
filter = 1/2 * (Gaussianfilter(ks,sigma,mu) + Gaussianfilter(ks,sigma,-mu));
subplot(3,1,2)
plot(ks,filter)
xlabel('k (wave number)')
ylabel('Wave Amplitude')
title('Mixed-Gaussian Filter (Norm 1)')
```

```
axis tight
subplot(3,1,3)
Y1tf = filter'.*Y1t;
plot(ks, abs(Y1tf))
xlabel('k (wave number)')
ylabel('Wave Amplitude')
title(sprintf('Filtered Frequency Domain Representation \n of 1st PC'))
axis tight
Y1f = ifft(ifftshift(Y1tf));
figure(1)
subplot(3,1,1)
hold on
plot(t,(Y1f-Y1f(1))/max(abs(Y1f(:))))
hold off

% V = axis;
% V(1:2) = [0 length(t)];
V = [0 length(t) -0.9 .9];
axis(V)

legend('Unfiltered','Filtered','Location','Best')

fig1 = figure(1);
sgtitle('Principle Components for Test 2')
subplot(3,1,2)
plot(t, Y(:,2))
V = axis;
V(1:2) = [0 length(t)];
axis(V)
ylabel('Displacment (px)')
xlabel('Video Frame (number)')
title('2nd PC')

subplot(3,1,3)
plot(t, Y(:,3))
V = axis;
V(1:2) = [0 length(t)];
axis(V)
ylabel('Displacment (px)')
xlabel('Video Frame (number)')
title('3rd PC')

fig2 = figure(2);
plot(1:length(lambda), lambda, '-o', 'MarkerEdgeColor','k',...
    'MarkerFaceColor',[0.75,0.75,1])
xticks(1:length(lambda))
ylabel('Variance')
xlabel('Principal Component')
title(sprintf('Magnitude of Variance Captured \n by each Principal Component \n (Test 2)'))

%% Saving

saveas(fig1,'PC2.png')
saveas(fig2,'V2.png')
```

```
%% (test 3) Horizontal Displacement
clear variables; close all; clc;
%% Cam 1_3

load cam1_3.mat

datasize = size(vidFrames1_3); % height, width, color channels, length

x_a = zeros([datasize(4) 1]); y_a = x_a;

for j=1:datasize(4)
    gryfrm = rgb2gray(vidFrames1_3(:,:,:,j));
    frm = gryfrm>=245;
    frm(1:200,:) = 0; % remove artifact from top
    frm(:,1:250) = 0; % remove artifact from left side
    frm(:,end-100:end) = 0; % remove artifact from left side
    %imshow(frm, []);
    %hold on
    [x_a(j), y_a(j)] = centroid(frm);
    %scatter(xcom, ycom, 50, 'x');
    %hold off
    %drawnow
end

clear vidFrames1_3;
'Cam1_3 done.'

%plot(y_a)

%% Cam 2_3

load cam2_3.mat;

x_b = zeros([datasize(4) 1]); y_b = x_b;

for j=1:datasize(4)
    gryfrm = rgb2gray(vidFrames2_3(:,:,:,j));
    frm = gryfrm>=250;
    frm(1:175,:) = 0; % remove artifact from top
    %imshow(frm, []);
    %hold on
    [x_b(j), y_b(j)] = centroid(frm);
    %scatter(xcom, ycom, 50, 'x');
    %hold off
    drawnow
end

clear vidFrames2_1;
'Cam2_3 done.'

%plot(y_b)

%% Cam 3_3
```

```
load cam3_3.mat;

datasize = size(vidFrames3_3); % height, width, color channels, length

x_c = zeros([datasize(4) 1]); y_c = x_c;

for j=1:datasize(4)
    gryfrm = rgb2gray(vidFrames3_3(:,:,:,j));
    frm = gryfrm>=245;
    frm(:,1:200) = 0; % remove artifact from left side
    %imshow(frm, []);
    %hold on
    [x_c(j), y_c(j)] = centroid(frm);
    %scatter(xcom, ycom, 50, 'x');
    %hold off
    drawnow
end

clear vidFrames3_1;
'Cam3_3 done.'

%plot(x_c)

%% (test 4) Horizontal Displacement Analysis

% Not all sensors started recording at the same time.
% We need to synchronize the recordings. This means we will have
% to synchronize the 1st shift in direction and trim frames not shared
% by all videos.

t_a = 1:length(x_a);
t_b = 1:length(x_b);
t_c = 1:length(x_c);

figure(3)
subplot(3,1,1)
[pksa, loca] = findpeaks(y_a, 'MinPeakDistance', 35);
findpeaks(y_a, 'MinPeakDistance', 35)
subplot(3,1,2)
[pksb, locb] = findpeaks(y_b, 'MinPeakDistance', 35); % 2nd peak
findpeaks(y_b, 'MinPeakDistance', 35)
subplot(3,1,3)
[pksc, locc] = findpeaks(x_c, 'MinPeakDistance', 35);
findpeaks(x_c, 'MinPeakDistance', 35)

syncpeaks = [loca(2) locb(2) locc(2)];

chopinit = syncpeaks - min(syncpeaks); %
achop = 1:chopinit(1);
x_a(achop) = []; y_a(achop) = []; t_a(achop) = [];
bchop = 1:chopinit(2);
x_b(bchop) = []; y_b(bchop) = []; t_b(bchop) = [];
cchop = 1:chopinit(3);
x_c(cchop) = []; y_c(cchop) = []; t_c(cchop) = [];
```

```
maxvidlength = min([length(t_a) length(t_b) length(t_c)]);
t = 1:maxvidlength;

x_a = x_a(t); x_b = x_b(t); x_c = x_c(t);
y_a = y_a(t); y_b = y_b(t); y_c = y_c(t);

figure(4)
subplot(3,1,1)
plot(t,y_a)
subplot(3,1,2)
plot(t,y_b)
subplot(3,1,3)
plot(t,x_c)

close all;

X = [x_a'
    y_a'
    x_b'
    y_b'
    x_c'
    y_c'];

% Code from the book:
[m,n]=size(X); % compute data size
[u,s,v]=svd((X-mean(X,2))/sqrt(n-1)); % perform the SVD
lambda=diag(s).^2; % produce diagonal variances
Y=u'*X; % produce the principal components projection

fig1 = figure(1);
sgtitle('Principle Components for Test 3')
subplot(3,1,1)
Y = Y';
plot(t, Y(:,1))
title('1st PC')
ylabel('Displacment (px)')
xlabel('Video Frame (number)')
V = axis;
V(1:2) = [0 length(t)];
axis(V)

subplot(3,1,2)
plot(t, Y(:,2))
V = axis;
V(1:2) = [0 length(t)];
axis(V)
ylabel('Displacment (px)')
xlabel('Video Frame (number)')
title('2nd PC')

subplot(3,1,3)
plot(t, Y(:,3))
V = axis;
```

```
V(1:2) = [0 length(t)];
axis(V)
ylabel('Displacment (px)')
xlabel('Video Frame (number)')
title('3rd PC')

fig2 = figure(2);
plot(1:length(lambda), lambda, '-o', 'MarkerEdgeColor','k',...
    'MarkerFaceColor',[0.75,0.75,1])
xticks(1:length(lambda))
ylabel('Variance')
xlabel('Principal Component')
title(sprintf('Magnitude of Variance Captured \n by each Principal Component \n (Test 3)'))

%% Saving

saveas(fig1,'PC3.png')
saveas(fig2,'V3.png')

%% (test 4) Horizontal Displacement and Rotation
clear variables; close all; clc;
%% Cam 1_4

load cam1_4.mat

datasize = size(vidFrames1_4); % height, width, color channels, length

x_a = zeros([datasize(4) 1]); y_a = x_a;

for j=1:datasize(4)
    gryfrm = rgb2gray(vidFrames1_4(:,:,:,j));
    frm = gryfrm>=240;
    frm(1:175,:) = 0; % remove artifact
    frm(:,1:275) = 0; % remove artifact
    %imshow(frm, []);
    %hold on
    [x_a(j), y_a(j)] = centroid(frm);
    %scatter(xcom, ycom, 50, 'x');
    %hold off
    drawnow
end

clear vidFrames1_4;
'Cam1_4 done.'

%plot(y_a)

%% Cam 2_4

load cam2_4.mat;

x_b = zeros([datasize(4) 1]); y_b = x_b;

for j=1:datasize(4)
```

```
    gryfrm = rgb2gray(vidFrames2_4(:,:,:,j));
    frm = gryfrm>=245;
    frm(:,[1:205 end-225:end]) = 0; % remove artifact from sides
    frm([end-90:end],:) = 0; % remove artifact from bottom
    %imshow(frm, []);
    %hold on
    [x_b(j), y_b(j)] = centroid(frm);
    %scatter(xcom, ycom, 50, 'x');
    %hold off
    drawnow
end

clear vidFrames2_4;
'Cam2_4 done.'

%plot(y_b)

%% Cam 3_4
load cam3_4.mat;

x_c = zeros([datasize(4) 1]); y_c = x_c;

for j=1:datasize(4)
    gryfrm = rgb2gray(vidFrames3_4(:,:,:,j));
    frm = gryfrm>=235;
    frm(:,1:225) = 0; % remove artifact from left
    %imshow(frm, []);
    %hold on
    [x_c(j), y_c(j)] = centroid(frm);
    %scatter(xcom, ycom, 50, 'x');
    %hold off
    drawnow
end

clear vidFrames3_4;
'Cam3_4 done.'

%plot(x_c)

%% (test 4) Horizontal Displacement and Rotation Analysis

% Not all sensors started recording at the same time.
% We need to synchronize the recordings. This means we will have
% to synchronize the 1st shift in direction and trim frames not shared
% by all videos.

t_a = 1:length(x_a);
t_b = 1:length(x_b);
t_c = 1:length(x_c);

figure(3)
subplot(3,1,1)
[pksa, loca] = findpeaks(y_a, 'MinPeakDistance', 35);
findpeaks(y_a, 'MinPeakDistance', 35)
```

```
subplot(3,1,2)
[pksb, locb] = findpeaks(y_b, 'MinPeakDistance', 35); % 2nd peak
findpeaks(y_b, 'MinPeakDistance', 35)
subplot(3,1,3)
[pksc, locc] = findpeaks(x_c, 'MinPeakDistance', 35);
findpeaks(x_c, 'MinPeakDistance', 35)

syncpeaks = [loca(2) locb(2) locc(2)];

chopinit = syncpeaks - min(syncpeaks); %
achop = 1:chopinit(1);
x_a(achop) = []; y_a(achop) = []; t_a(achop) = [];
bchop = 1:chopinit(2);
x_b(bchop) = []; y_b(bchop) = []; t_b(bchop) = [];
cchop = 1:chopinit(3);
x_c(cchop) = []; y_c(cchop) = []; t_c(cchop) = [];

maxvidlength = min([length(t_a) length(t_b) length(t_c)]);
t = 1:maxvidlength;

x_a = x_a(t); x_b = x_b(t); x_c = x_c(t);
y_a = y_a(t); y_b = y_b(t); y_c = y_c(t);

figure(4)
subplot(3,1,1)
plot(t,y_a)
subplot(3,1,2)
plot(t,y_b)
subplot(3,1,3)
plot(t,x_c)

close all;

X = [x_a'
    y_a'
    x_b'
    y_b'
    x_c'
    y_c'];

% Code from the book:
[m,n]=size(X); % compute data size
[u,s,v]=svd((X-mean(X,2))/sqrt(n-1)); % perform the SVD
lambda=diag(s).^2; % produce diagonal variances
Y=u'*X; % produce the principal components projection

fig1 = figure(1);
sgtitle('Principle Components for Test 4')
subplot(4,1,1)
Y = Y';
plot(t, Y(:,1))
title('1st PC')
ylabel('Displacment (px)')
xlabel('Video Frame (number)')
```

```
V = axis;
V(1:2) = [0 length(t)];
axis(V)

subplot(4,1,2)
plot(t, Y(:,2))
V = axis;
V(1:2) = [0 length(t)];
axis(V)
ylabel('Displacment (px)')
xlabel('Video Frame (number)')
title('2nd PC')

subplot(4,1,3)
plot(t, Y(:,3))
V = axis;
V(1:2) = [0 length(t)];
axis(V)
ylabel('Displacment (px)')
xlabel('Video Frame (number)')
title('3rd PC')

subplot(4,1,4)
plot(t, Y(:,4))
V = axis;
V(1:2) = [0 length(t)];
axis(V)
ylabel('Displacment (px)')
xlabel('Video Frame (number)')
title('4th PC')

fig2 = figure(2);
plot(1:length(lambda), lambda, '-o', 'MarkerEdgeColor','k',...
    'MarkerFaceColor',[0.75,0.75,1])
xticks(1:length(lambda))
ylabel('Variance')
xlabel('Principal Component')
title(sprintf('Magnitude of Variance Captured \n by each Principal Component \n (Test 4)'))

%% Saving

saveas(fig1,'PC4.png')
saveas(fig2,'V4.png')
```