Homework 2: PCA
AMATH 482: Computational Methods for Data Analysis, Winter 2017

Sean Lam
seanlam@uw.edu

February 6, 2017

Abstract

This report investigates the use of principal component analysis (PCA) to empirically extract the governing equations of the motion of a spring-mass system. Multiple cameras were used to record the system at different angles and positions to capture the full dynamics of the system. The role of principal component analysis was to produce low-dimensional reductions of the dynamics and behaviors of the system from noisy, seemingly complex and random data.

I. Introduction and Overview

We live in a world where data is being collected everywhere, and it is very easy to do so. Principal component analysis is a great linear algebra tool to find strong patterns in a data set in the midst of seemingly complex, random, and noisy data. This method can be used to quantify low dimensional dynamics in complex systems. Some examples of applications are turbulent fluid flow, structural vibrations, and computational neuroscience.

In this experiment, principal component analysis was used to empirically determine the dynamics of a spring-mass system. A paint can was attached to a spring and was released and held by a student while three video cameras were recording its motion at different angles and positions. A small flashlight was placed on top of the paint can to make motion tracking more convenient. This experiment was done a total of four times. In Test 1, the spring was released as straight as possible so most of the motion was in the z direction. In Test 2, the spring was released ideally again, but the cameras were shaking a lot, creating noisy data. In Test 3, the spring was released with horizontal displacement. In Test 4, the spring was released with both horizontal displacement and rotation. The goal of this assignment was to see if the results for the ideal case could be reproduced from the noisy cases.

II. Theoretical Background

A simple way to identify redundant data in is by analyzing the covariance between data sets. Strongly statistically dependent variables can be classified as redundancies in the data set.

Consider two data sets in the form of row vectors with each having a mean of zero.

$$\mathbf{a} = \begin{bmatrix} a_1 & a_2 & ... & a_n \end{bmatrix} \tag{1}$$

$$\mathbf{b} = \begin{bmatrix} b_1 & b_2 & ... & b_n \end{bmatrix} \tag{2}$$

The variances of a and b are given by

$$\sigma_{\mathbf{a}}^{\mathbf{2}} = \frac{1}{n-1}\mathbf{a}\mathbf{a}^T \tag{3}$$

$$\sigma_{\mathbf{b}}^{\mathbf{2}} = \frac{1}{n-1}\mathbf{b}\mathbf{b}^T \tag{4}$$

The covariance between the two data sets is given by

$$\sigma_{\mathbf{ab}}^{\mathbf{2}} = \frac{1}{n-1}\mathbf{a}\mathbf{b}^T \tag{5}$$

For this experiment, we have a 6 row data set $\mathbf{X}$ (3 sets of X and Y coordinates). The covariance matrix for this data set is

$$\mathbf{C_x} = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T \tag{6}$$

The covariance matrix is a square, symmetric $m \times m$ matrix whose diagonal represents the variance of particular measurements. The off-diagonal terms are the covariances between measurement types. Large diagonal terms normally represent the dynamics of interest in a system since the large variance suggests strong fluctuations in that variable. Redundancy can be determined if an off-diagonal term and diagonal term are equal since $\sigma^{\mathbf{2}}_{\mathbf{ab}} = \sigma^{\mathbf{2}}_{\mathbf{a}} = \sigma^{\mathbf{2}}_{\mathbf{b}}$ if $\mathbf{a} = \mathbf{b}$.

What the PCA does is that it diagonalizes the covariance matrix, so the diagonals are ordered from largest to smallest and the off-diagonals are zero. This is how it is used for dimensional reduction.

The singular value decomposition decomposes a matrix into three constitutive components,

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{7}$$

where $\mathbf{U}$ and $\mathbf{V}$ are unitary orthogonal bases, and $\mathbf{\Sigma}$ is a diagonal matrix. To diagonalize the covariance matrix with the SVD method, we first define the transformed variable

$$\mathbf{Y} = \mathbf{U}^*\mathbf{X} \tag{8}$$

The covariance of this transformed variable is then

$$\begin{aligned}\mathbf{C_Y} &= \frac{1}{n-1}\mathbf{Y}\mathbf{Y}^T \\ &= \frac{1}{n-1}(\mathbf{U}^*\mathbf{X})(\mathbf{U}^*\mathbf{X})^T \\ &= \frac{1}{n-1}\mathbf{U}^*(\mathbf{X}\mathbf{X}^T)\mathbf{U} \\ &= \frac{1}{n-1}\mathbf{U}^*\mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^*\mathbf{U}\end{aligned}$$

$$\mathbf{C_Y} = \frac{1}{n-1}\mathbf{\Sigma^2} \tag{9}$$

III. Algorithm Implementation and Development

Twelve videos were given in total: four tests with three videos each. The videos were colored and were downloaded in the form of a four dimensional matrix, with the pixel dimensions as the first two dimensions, RGB as the third dimension, and video frame as the fourth dimension. For simplicity, the first step was to convert all of the colored videos into grayscale.

Next, all of the videos were manually cropped to eliminate as much of the background as possible to keep the focus on the paint can. This helps with the accuracy of the motion tracking algorithm because the algorithm recognizes the paint can as the brightest pixel intensities, so cropping the video eliminates all other bright spots in the videos, such as lights, the glare on the whiteboard, and the glare on the person's watch. The videos were also rotated if necessary so that the orientation of the room would be the same each video (the floor is parallel to top and bottom edge of video).

After cropping and rotating the video, the paint can was isolated from the background by changing all pixels under 252 intensity to 0 (black) since there was a flashlight on top of the paint can, which made it easier to track the system because then I could track the highest (in terms of position) group of pixels and that would be essentially tracking the flashlight. For each frame, the algorithm started at the top row and iterated down the rows, until it found a row with at least one bright pixel. It was assumed that that row was the top of the flashlight. The algorithm then iterated down rows again until there were at least ten consecutive rows without a bright pixel. It was assumed that at that point, the entirety of the flashlight pixels were collected. The algorithm then calculated the average position out of all the pixel positions within the region. This was done for every frame. There are a few exceptions to this where the flashlight would often be pointing away from the camera, so tracking the highest group of bright pixels was not very accurate. However, the bottom of the paint can was also very bright, because the paint can was white and there was glare on the bottom of the paint can, so an alternative algorithm was used in these cases. The alternative algorithm does the same thing as the original one except it finds the lowest group of high intensity pixels, and iterates upwards. This was not as effective as tracking the flashlight because the glare was what made the bottom of the paint can so bright, and the glare changes a little bit depending on the position of the paint can. However, it is consistent enough, as seen in figures 1-4.

After collecting all the positions, I had to manually perform phase shifts on the videos because not all of the cameras started recording at the same time. I shifted all of the videos so that they would start when the paint can was at it's highest point. Then, I truncated all the videos so that in each test, the videos would have the same number of frames.

The data matrix was organized into the following matrix

$$\mathbf{X} = \begin{bmatrix} x_1 & \dots\dots.. \\ y_1 & \dots\dots.. \\ x_2 & \dots\dots.. \\ y_2 & \dots\dots.. \\ x_3 & \dots\dots.. \\ y_3 & \dots\dots.. \end{bmatrix} \tag{10}$$

The last step is to normalize all of the data so that the y direction would have a mean of zero and a standard deviation of one. Each x direction was also normalized to be centered at zero, but then was divided by the standard deviation of its respective y position data. This is equivalent to scaling each video, because the cameras were at different distances away from the paint can. If the camera was far away from the paint can, it would not travel along many pixels (low amplitude), whereas if the camera was really close to the paint can, it would seem to have a very high amplitude.

After all the data has been normalized, the last step was to take the singular value decomposition of the positions and then perform the calculations in Section II to find the covariances and determine the dynamics of the spring-mass system.

IV. Computational Results

The first step of the analysis was to first confirm that the data was collected properly by checking Y positions had an oscillatory behavior, because the dynamics of a spring were already known, the purpose of this experiment was to replicate them. Figure 1 shows the normalized and truncated positions for Test 1, the ideal case. All cameras produced basically the same motion in the Y direction, and the X direction was has a similar trend on all of the cameras.

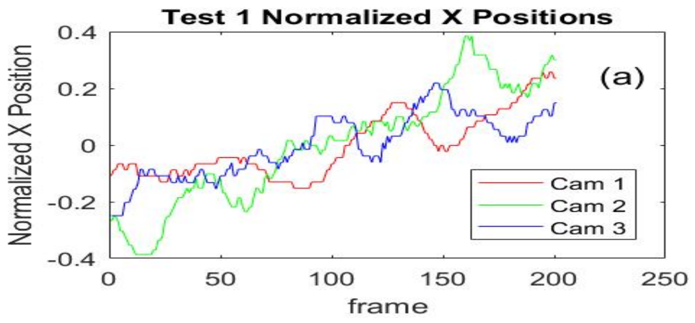
Figure 5a shows the variance in each mode for Test 1. The first principal component has over 96% of the total variance, which was expected because Test 1 was the ideal case. Figure 5b shows the first three principal components, and it shows that mode 1 captures basically all the oscillatory behavior in the system.

Figure 2 shows the positions for Test 2, the noisy case with camera shake. The algorithm was still able to successfully track the paint can for the most part but because of the camera shake, the positions plotted on the graph did not capture the "true" motion because if the camera moves, it appears that the paint can is moving even if it isn't. However, it is still clear that there is oscillatory behavior on the y direction for test 2.

Figure 6a shows the variance in each mode for Test 2. The first principal component still dominates with about 64% of the total variance, and the second principal component has about 15%. The results are significantly different than for test 1 because of all the noise from the camera shake. Figure 6b shows the first three principal components. The first mode still captures most of the oscillatory behavior in the system, while the second and third modes look more random. These probably capture all of the camera shake.

Figure 3 shows the positions for Test 3, the case with horizontal displacement (pendulum motion). The Y direction was pretty consistent throughout the three cameras. There is some error due to the light disappearing for short amounts of time because it was facing away from the camera. Figure 3a shows the pendulum oscillatory motion on the X direction, which is what was expected.

Figure 7a shows the variance in each mode for Test 3. The first and second principal components include almost 90% of the total variance, which are better results than Test 2. As shown in Figure 7b, the first three principal components all capture oscillatory behavior. This is interesting because for all the other tests, only the first principal components capture oscillatory behavior, while the rest just capture noisy and random data. This



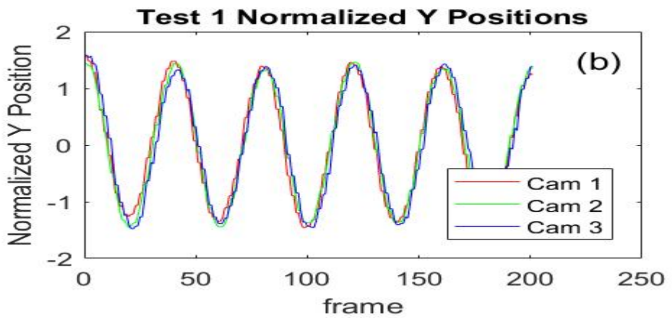
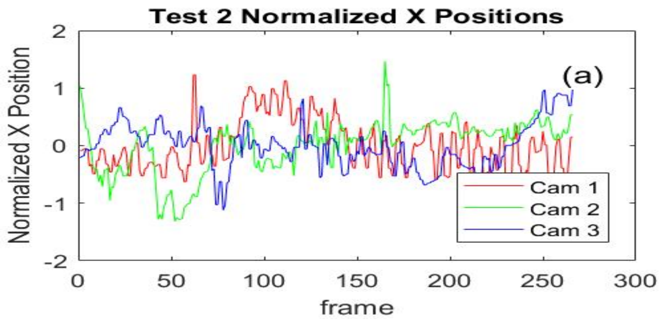


Figure 1: These are the normalized positions for Test 1, the ideal case. a) are the X positions, b) are the Y positions.



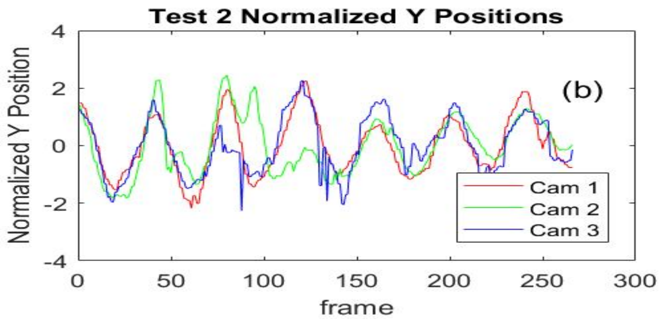


Figure 2: These are the normalized positions for Test 2, the noisy case (Camera shake). a) are the X positions, b) are the Y positions.

is because both the X and Y directions contained oscillatory behavior: the Y direction had simple harmonic oscillations and the X direction had a pendulum motion.

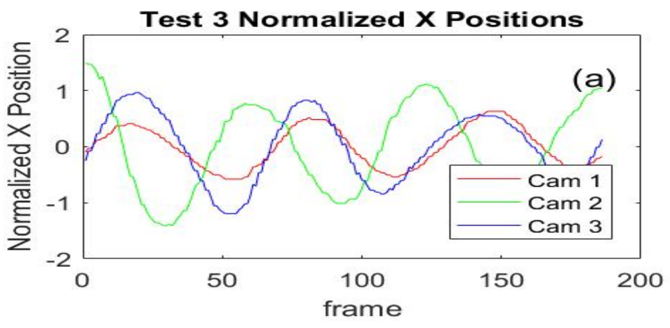
Figure 4 shows the positions for Test 4, the case with both pendulum motion and rotation. From looking at the Y direction, it is still clear that the paint can had oscillatory behavior, and the cameras are for the most part pretty consistent. Again, there is some error due to the light disappearing for short amounts of time because the rotation caused it to face away from the camera. The X direction seemed a little more random, as seen in Figure 4a, which is again due to the inconsistencies with the light facing away.

Figure 8a shows the variance in each mode for Test 4. The first principal component dominates with 76% of the total variance, and the first three principal components capture about 93% of the total variance. These results are similar to Test 3, the only difference between Tests 3 and 4 is that Test 4 added rotation. In test 3, we could clearly see the principal components for oscillatory behavior in both the X and Y directions, but in figure 7b, only the first mode is clearly oscillating. This is because rotation and horizontal displacement both affected the X direction, and the two combined together must have made the data more convoluted. This is why the first principal component has a higher percent of the total variance for Test 4 than in Test 3. It does not appear that the PCA was able to capture the horizontal displacement and rotation separately because we were tracking essentially a two dimensional video.

V. Summary and Conclusions

In this assignment, the PCA was used to empirically determine the dynamics of a paint can caught on video using three cameras. The PCA was able to capture the simple harmonic oscillatory behavior of the paint can for each of the four tests. It successfully performed low-dimensional reduction on the data since we oversampled the data by using three cameras capturing essentially the same motion. The PCA was also able to separate noisy data due to camera shakes, random horizontal displacement, and random rotation.

By performing principal component analysis to confirm the dynamics of a spring-mass system which was already known from Newtonian physics, this project shows how the PCA is a powerful linear algebra tool to



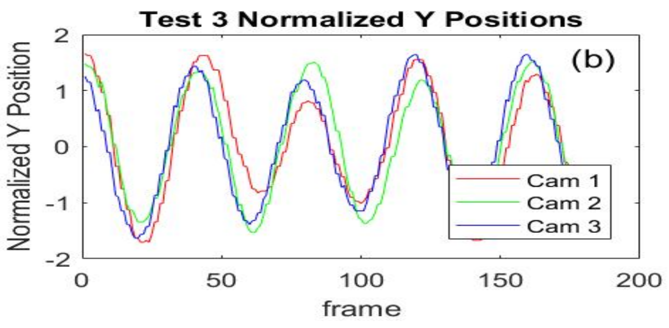
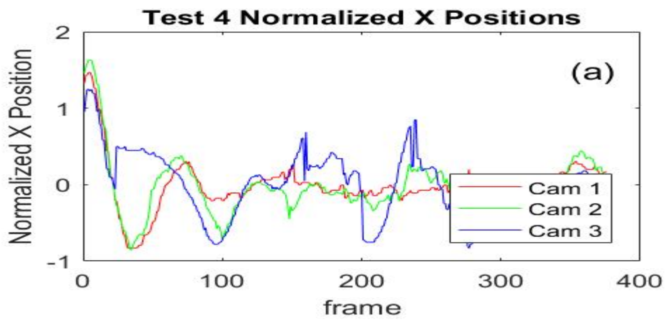


Figure 3: These are the normalized positions for Test 3, the case with horizontal displacement. a) are the X positions, b) are the Y positions.



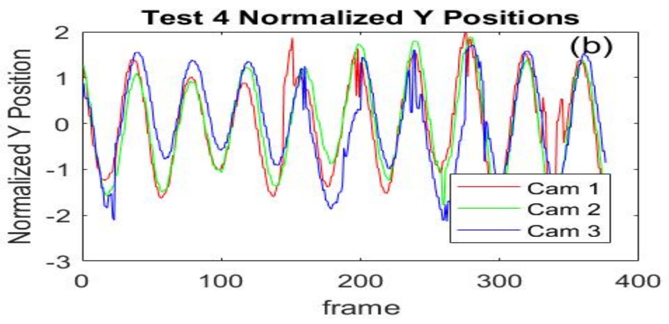
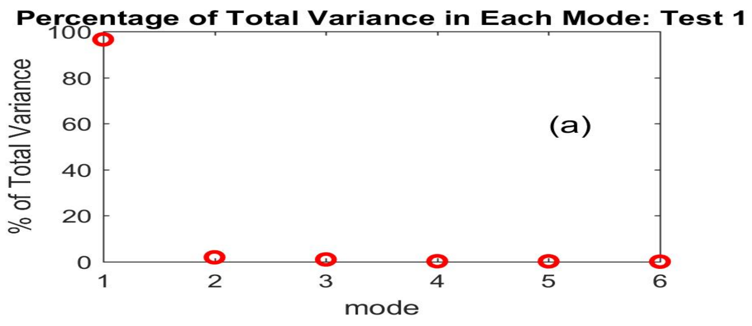


Figure 4: These are the normalized positions for Test 4, the case with horizontal displacement and rotation. a) are the X positions, b) are the Y positions.



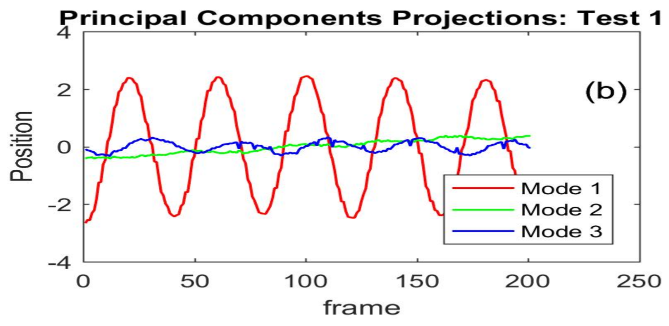


Figure 5: These are the results from the principal component analysis of Test 1. a) shows the variance of each mode, and b) shows the first three modes.

analyze large sets of data in almost every field. The applications are endless, ranging from economics, physical engineering and sciences, neuroscience, medicine, biology, signal processing, communications, and more.

Appendix A: MATLAB functions used and brief implementation explanation

svd(A) : performs singular value decomposition on matrix A and outputs U, Σ, and V
zscore(A) : normalizes matrix A and outputs matrix Y (the normalized matrix with each column having a mean of zero and standard deviation of one), μ (the average of each column of matrix A), and σ (the standard deviation of each column of matrix A)
imrotate(A, theta) : rotates image A by theta degrees in the counterclockwise direction
grayscale(A) : converts colored video A to a grayscale video

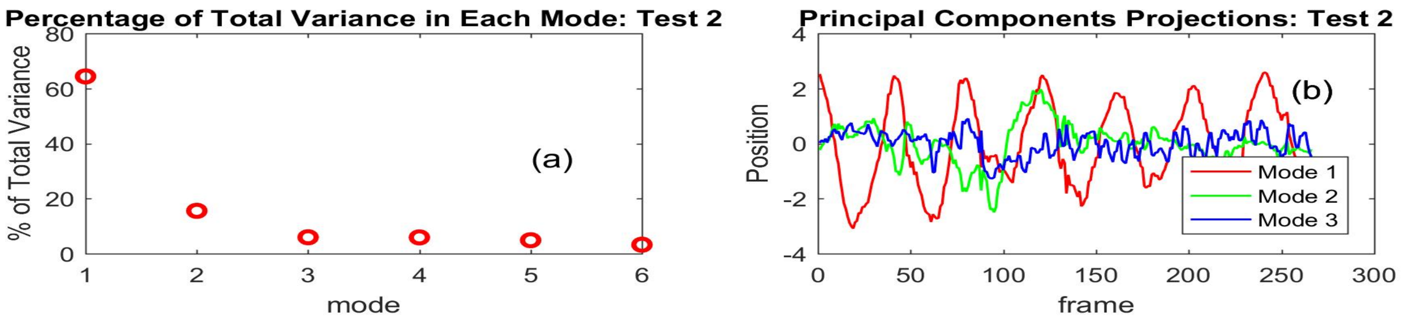


Figure 6: These are the results from the principal component analysis of Test 2. a) shows the variance of each mode, and b) shows the first three modes.

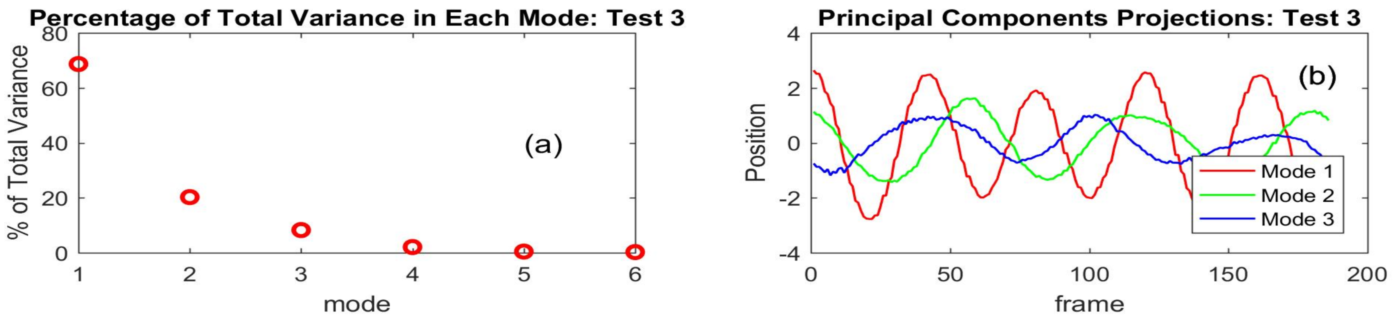


Figure 7: These are the results from the principal component analysis of Test 3. a) shows the variance of each mode, and b) shows the first three modes.

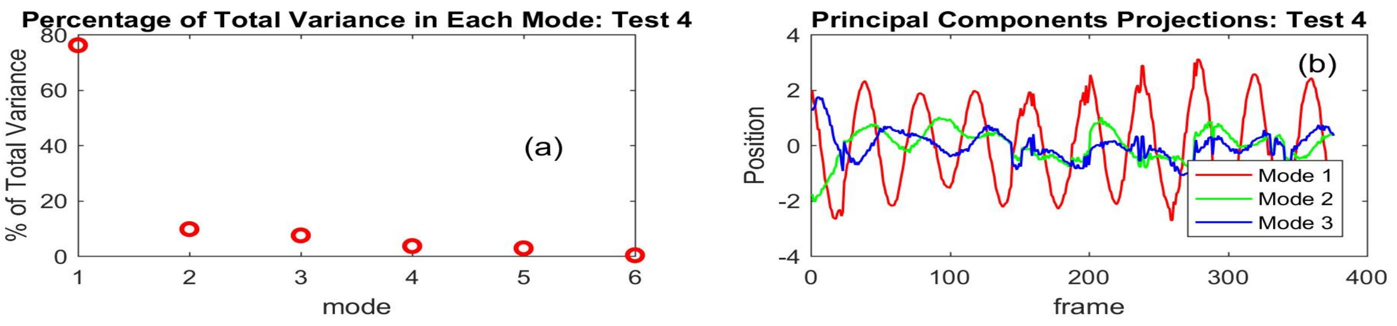


Figure 8: These are the results from the principal component analysis of Test 4. a) shows the variance of each mode, and b) shows the first three modes.

cropVideo(A, y0, x0, yf, xf) : crops video A starting from x0,y0 to xf,yf
rotateVideo(A, theta) : rotates video A by theta degrees in the counterclockwise direction
FindXYhighest(A, B) : tracks the highest group of bright pixels (group of pixels is determined by a group of pixels with no more than B number of rows of darkness in between)
FindXYlowest(A, B) : tracks the lowest group of bright pixels (group of pixels is determined by a group of pixels with no more than B number rows of darkness in between)
Normalize(A) : takes in a matrix A of XY positions, and normalizes the positions to have a mean of zero and standard deviation of one
PCACode(A): takes in a matrix A of XY positions, and performs principal component analysis, outputting $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}$, λ (the variances), $\mathbf{Y} = \mathbf{U}^*\mathbf{A}$, and $\mathbf{C_Y}$, the covariance of $\mathbf{Y}$

Appendix B: MATLAB codes

LoadAll.m

```
load cam1_1.mat
load cam2_1.mat
load cam3_1.mat
load cam1_2.mat
load cam2_2.mat
load cam3_2.mat
load cam1_3.mat
load cam2_3.mat
load cam3_3.mat
load cam1_4.mat
load cam2_4.mat
load cam3_4.mat
```

ProcessVideos.m

```
%% grayscale all videos
vid1_1 = grayscale(vidFrames1_1);
vid2_1 = grayscale(vidFrames2_1);
vid3_1 = grayscale(vidFrames3_1);
vid1_2 = grayscale(vidFrames1_2);
vid2_2 = grayscale(vidFrames2_2);
vid3_2 = grayscale(vidFrames3_2);
vid1_3 = grayscale(vidFrames1_3);
vid2_3 = grayscale(vidFrames2_3);
vid3_3 = grayscale(vidFrames3_3);
vid1_4 = grayscale(vidFrames1_4);
vid2_4 = grayscale(vidFrames2_4);
vid3_4 = grayscale(vidFrames3_4);

%% First Case
%First Vid
vid1_1crop = cropVideo(vid1_1, 150, 250, 450, 350);
%% Second Vid
vid2_1crop = cropVideo(vid2_1,75,250,350,350);

%% Third Vid
vid3_1rotate = rotateVideo(vid3_1,-80);
vid3_1crop = cropVideo(vid3_1rotate, 250, 200, 550, 350);

%% Second Case
%Vid 1
vid1_2crop = cropVideo(vid1_2,200,300,400,400);

%% Vid 2
vid2_2crop = cropVideo(vid2_2,1,150,480,500);

%% Vid 3
vid3_2rotate = rotateVideo(vid3_2,-80);
vid3_2crop = cropVideo(vid3_2rotate,300,225,500,350);

%% Third Case
%Vid 1
vid1_3crop = cropVideo(vid1_3, 200, 275, 425, 400);
```

```
%% Vid 2
vid2_3crop = cropVideo(vid2_3,100, 200, 400, 400);

%% Vid 3
vid3_3rotate = rotateVideo(vid3_3,-80);
vid3_3crop = cropVideo(vid3_3rotate, 300, 200, 500, 400);

%% Fourth Case
%Vid 1
vid1_4crop = cropVideo(vid1_4, 200, 300, 400, 475);

%% Vid 2
vid2_4crop = cropVideo(vid2_4,75,200,375,450);

%% Vid 3
vid3_4rotate = rotateVideo(vid3_4, -113);
vid3_4crop = cropVideo(vid3_4rotate, 350, 250, 600, 450);
```

FindAllPositions.m

```
%% Vid 1_1
[x1_1,y1_1,drawbox1_1] = FindXYhighest(vid1_1crop,10);

%% Vid 2_1
[x2_1,y2_1,drawbox2_1] = FindXYhighest(vid2_1crop,10);

%% Vid 3_1
[x3_1, y3_1, drawbox3_1] = FindXYhighest(vid3_1crop, 10);

%% Vid 1_2
[x1_2,y1_2,drawbox1_2] = FindXYhighest(vid1_2crop,10);

%% Vid 2_2
[x2_2,y2_2,drawbox2_2] = FindXYhighest(vid2_2crop,10);

%% Vid 3_2
[x3_2, y3_2, drawbox3_2] = FindXYhighest(vid3_2crop, 10);

%% Vid 1_3
[x1_3, y1_3, drawbox1_3] = FindXYlowest(vid1_3crop, 10);

%% Vid 2_3
[x2_3, y2_3, drawbox2_3] = FindXYlowest(vid2_3crop, 10);

%% Vid 3_3
[x3_3, y3_3, drawbox3_3] = FindXYhighest(vid3_3crop,10);

%% Vid 1_4
[x1_4, y1_4, drawbox1_4] = FindXYlowest(vid1_4crop, 10);

%% Vid 2_4
[x2_4, y2_4, drawbox2_4] = FindXYlowest(vid2_4crop, 10);

%% Vid 3_4
[x3_4, y3_4, drawbox3_4] = FindXYhighest(vid3_4crop,10);
```

```
%% Make XY matrix
truncate1 = 200;
start1_1 = 11;
start2_1 = 20;
start3_1 = 9;
Positions1 = [x1_1(1, start1_1: start1_1 + truncate1);
    y1_1(1, start1_1 : start1_1 + truncate1);
    x2_1(1, start2_1:start2_1 + truncate1);
    y2_1(1, start2_1:start2_1 + truncate1);
    x3_1(1, start3_1:start3_1+ truncate1);
    y3_1(1, start3_1:start3_1+truncate1); ];
%% Normalize Test 1 Data
Norm1 = Normalize(Positions1);

%%
truncate2 = 265;
start1_2 = 35;
start2_2 = 18;
start3_2 = 39;
Positions2 = [x1_2(start1_2:start1_2+truncate2);
    y1_2(start1_2:start1_2+truncate2);
    x2_2(start2_2:start2_2+truncate2);
    y2_2(start2_2:start2_2+truncate2);
    x3_2(start3_2:start3_2+truncate2);
    y3_2(start3_2:start3_2+truncate2); ];
Norm2 = Normalize(Positions2);

%%
truncate3 = 185;
start1_3 = 38;
start2_3 = 25;
start3_3 = 32;
Positions3 = [x1_3(start1_3:start1_3+truncate3);
    y1_3(start1_3:start1_3+truncate3);
    x2_3(start2_3:start2_3+truncate3);
    y2_3(start2_3:start2_3+truncate3);
    x3_3(start3_3:start3_3+truncate3);
    y3_3(start3_3:start3_3+truncate3); ];
Norm3 = Normalize(Positions3);

%%
truncate4 = 375;
start1_4 = 17;
start2_4 = 22;
start3_4 = 13;
Positions4 = [x1_4(start1_4:start1_4+truncate4);
    y1_4(start1_4:start1_4+truncate4);
    x2_4(start2_4:start2_4+truncate4);
    y2_4(start2_4:start2_4+truncate4);
    x3_4(start3_4:start3_4+truncate4);
    y3_4(start3_4:start3_4+truncate4); ];
Norm4 = Normalize(Positions4);
%% Plot for Test 1
colors = ['r', 'r', 'g', 'g', 'b', 'b'];
```

```
    subplot(1,2,1)
    for j = [1 3 5] %x positions
        plot(Norm1(j,:),colors(j));
        xlabel('frame')
        ylabel('Normalized X Position')
        hold on
    end
    title('Test 1 Normalized X Positions')
    text(220,.25,'(a)','Fontsize',[13])
    legend('Cam 1','Cam 2', 'Cam 3','Location','Southeast')
    subplot(1,2,2)
    for j = [2 4 6]
        plot(Norm1(j,:),colors(j));
        xlabel('frame')
        ylabel('Normalized Y Position')
        hold on
    end
    title('Test 1 Normalized Y Positions')
    text(220,1.5,'(b)','Fontsize',[13])
    legend('Cam 1','Cam 2', 'Cam 3','Location','Southeast')

    %% Plot for Test 2
    colors = ['r', 'r', 'g', 'g', 'b', 'b'];
    subplot(1,2,1)
    for j = [1 3 5] %x positions
        plot(Norm2(j,:),colors(j));
        xlabel('frame')
        ylabel('Normalized X Position')
        hold on
    end
    title('Test 2 Normalized X Positions')
    text(260,1.25,'(a)','Fontsize',[13])
    legend('Cam 1','Cam 2', 'Cam 3','Location','Southeast')
    subplot(1,2,2)
    for j = [2 4 6]
        plot(Norm2(j,:),colors(j));
        xlabel('frame')
        ylabel('Normalized Y Position')
        hold on
    end
    title('Test 2 Normalized Y Positions')
    text(260,2,'(b)','Fontsize',[13])
    legend('Cam 1','Cam 2', 'Cam 3','Location','Southeast')

%% Plot for Test 3
colors = ['r', 'r', 'g', 'g', 'b', 'b'];
    subplot(1,2,1)
    for j = [1 3 5] %x positions
        plot(Norm3(j,:),colors(j));
        xlabel('frame')
        ylabel('Normalized X Position')
        hold on
    end
    title('Test 3 Normalized X Positions')
    legend('Cam 1','Cam 2', 'Cam 3','Location','Southeast')
    text(175,1.25,'(a)','Fontsize',[13])
    subplot(1,2,2)
    for j = [2 4 6]
        plot(Norm3(j,:),colors(j));
        xlabel('frame')
        ylabel('Normalized Y Position')
        hold on
    end
    title('Test 3 Normalized Y Positions')
    text(175,1.6,'(b)','Fontsize',[13])
    legend('Cam 1','Cam 2', 'Cam 3','Location','Southeast')

%% Plot for Test 4
colors = ['r', 'r', 'g', 'g', 'b', 'b'];
    subplot(1,2,1)
    for j = [1 3 5] %x positions
        plot(Norm4(j,:),colors(j));
        xlabel('frame')
        ylabel('Normalized X Position')
        hold on
    end
    title('Test 4 Normalized X Positions')
    legend('Cam 1','Cam 2', 'Cam 3','Location','Southeast')
    text(350,1.5,'(a)','Fontsize',[13])
    subplot(1,2,2)
    for j = [2 4 6]
        plot(Norm4(j,:),colors(j));
        xlabel('frame')
        ylabel('Normalized Y Position')
        hold on
    end
    title('Test 4 Normalized Y Positions')
    legend('Cam 1','Cam 2', 'Cam 3','Location','Southeast')
    text(350,1.75,'(b)','Fontsize',[13])
```

Analysis.m

```
%% Test 1
[U1, S1, V1, lambda1, PCA1, covariance1] = PCAcode(Norm1);
figure(1)
subplot(1,2,1)
plot(100 * lambda1 / sum(lambda1),'or','Linewidth',[2])
xlabel('mode')
ylabel('% of Total Variance')
title('Percentage of Total Variance in Each Mode: Test 1')
text(5,60,'(a)','Fontsize',[13])
subplot(1,2,2)
colors = ['r', 'g', 'b',];
for j = 1:3
   plot(PCA1(j,:),colors(j),'Linewidth',[1])
   hold on
end
legend('Mode 1', 'Mode 2', 'Mode 3', 'Location', 'Southeast')
title('Principal Components Projections: Test 1')
xlabel('frame')
ylabel('Position')
```

```
text(225,2,'(b)','Fontsize',[13])
%% Test 2
[U2, S2, V2, lambda2, PCA2, covariance2] = PCAcode(Norm2);
figure(2)
subplot(1,2,1)
plot(100 * lambda2 / sum(lambda2),'or','Linewidth',[2])
xlabel('mode')
ylabel('% of Total Variance')
title('Percentage of Total Variance in Each Mode: Test 2')
text(5,35,'(a)','Fontsize',[13])
subplot(1,2,2)
colors = ['r', 'g', 'b',];
for j = 1:3
   plot(PCA2(j,:),colors(j),'Linewidth',[1])
   hold on
end
legend('Mode 1', 'Mode 2', 'Mode 3', 'Location', 'Southeast')
title('Principal Components Projections: Test 2')
xlabel('frame')
ylabel('Position')
text(255,2,'(b)','Fontsize',[13])
%% Test 3
[U3, S3, V3, lambda3, PCA3, covariance3] = PCAcode(Norm3);
figure(3)
subplot(1,2,1)
plot(100 * lambda3 / sum(lambda3),'or','Linewidth',[2])
xlabel('mode')
ylabel('% of Total Variance')
title('Percentage of Total Variance in Each Mode: Test 3')
text(5,40,'(a)','Fontsize',[13])
subplot(1,2,2)
colors = ['r', 'g', 'b',];
for j = 1:3
   plot(PCA3(j,:),colors(j),'Linewidth',[1])
   hold on
end
legend('Mode 1', 'Mode 2', 'Mode 3', 'Location', 'Southeast')
title('Principal Components Projections: Test 3')
xlabel('frame')
ylabel('Position')
text(175,2.5,'(b)','Fontsize',[13])
%% Test 4
[U4, S4, V4, lambda4, PCA4, covariance4] = PCAcode(Norm4);
figure(4)
subplot(1,2,1)
plot(100 * lambda4 / sum(lambda4),'or','Linewidth',[2])
xlabel('mode')
ylabel('% of Total Variance')
title('Percentage of Total Variance in Each Mode: Test 4')
text(5,40,'(a)','Fontsize',[13])
subplot(1,2,2)
colors = ['r', 'g', 'b',];
for j = 1:3
   plot(PCA4(j,:),colors(j),'Linewidth',[1])
   hold on
```

```
end
legend('Mode 1', 'Mode 2', 'Mode 3', 'Location', 'Southeast')
title('Principal Components Projections: Test 4')
xlabel('frame')
ylabel('Position')
text(350,3,'(b)','Fontsize',[13])
```

grayscale.m

```
%turns RGB video into grayscale
function grayVideo = Grayscale(video)
    [~,~,~,frames] = size(video);
    for j = 1:frames
       grayFrame = rgb2gray(video(:,:,:,j));
       grayVideo(:,:,j) = grayFrame(:,:);
    end
end
```

cropVideo.m

```
%crops a video into x by y pixels starting at x0, y0 to xf, yf
function croppedVideo = cropVideo(video, y0, x0, yf, xf)
    [~,~,frames] = size(video);
    for j = 1:frames
       croppedVideo(:,:,j) = video(y0:yf, x0:xf, j);
    end
end
```

rotateVideo.m

```
%rotates video by angle in degrees CCW
function rotatedVideo = rotateVideo(video, angle)
    [~,~,frames] = size(video);
    for j = 1:frames
       rotatedVideo(:,:,j) = imrotate(video(:,:,j),angle);
    end
end
$
```

FindXYhighest

```
%this one finds x, y by just searching for the HIGHEST bright pixel
%also outputs a new video with a box around region of collected pixels
function [x, y, drawbox] = FindXYhighest(video, boundary)
    clear x; clear y;
    video = BrightestPixels(video);
    drawbox = video;
    [rows, cols, frames] = size(video);
    %for each frame
    for ii = 1:frames
       %find highest brightest pixel
       found = 0;
       ymax = 1;
       xsum = 0;
       ysum = 0;
       pixelcount = 0;
       %while haven't found pixel
       while (found == 0)
           if (max(video(ymax,:,ii)) ~= 0) %pixel found
```

```
                found = 1;
            else
               ymax = ymax + 1; %go to next row
            end
        end
        %keep going down rows until there are no more pixels for boundary rows
        %that's when you know you have found all of the light
        nolight = 0;
        currentrow = ymax;
        minx = 1000;
        maxx = 0;
        while (nolight < boundary) %while we haven't hit boundary blank rows
            %check if row is empty
            if (max(video(currentrow,:,ii)) == 0)
                %if it's empty then just add to nolight
                nolight = nolight + 1;
            else
                nolight = 0;
                %get all positions of x and y
                for j = 1:cols
                   if (video(currentrow,j, ii) ~= 0) %if is bright
                      minx = min(minx, j);
                      maxx = max(maxx, j);
                      xsum = xsum + j;
                      ysum = ysum + currentrow;
                      pixelcount = pixelcount + 1;
                   end
                end
            end
            %next row
            currentrow = currentrow + 1;
        end
        x(ii) = round(xsum / pixelcount, 0);
        y(ii) = round(ysum / pixelcount, 0);
        drawbox(ymax,minx:maxx,ii) = 255;
        drawbox(currentrow,minx:maxx,ii) = 255;
        drawbox(ymax:currentrow,minx,ii) = 255;
        drawbox(ymax:currentrow, maxx, ii) = 255;
    end

    %flip y coordinate to switch from pixel to xy coordinate system
    y = -1 .* y;
end
```

FindXYlowest

```
%this one finds xy by just searching for the LOWEST bright pixel
%also outputs a new video with a box around region of collected pixels
function [x, y, drawbox] = FindXYlowest(video, boundary)
    clear x; clear y;
    video = BrightestPixels(video);
    drawbox = video;
    [rows, cols, frames] = size(video);
    %for each frame
    for ii = 1:frames
       %find lowest brightest pixel
       found = 0;
       ymax = rows;
       xsum = 0;
       ysum = 0;
       pixelcount = 0;
       %while haven't found pixel
       while (found == 0)
           if (max(video(ymax,:,ii)) ~= 0) %pixel found
               found = 1;
           else
              ymax = ymax - 1; %go to next row
           end
       end
       %keep going up rows until there are no more pixels for boundary rows
       %that's when you know you have found all of the light
       nolight = 0;
       currentrow = ymax;
       minx = 1000;
       maxx = 0;
       while (nolight < boundary) %while we haven't hit boundary blank rows
           %check if row is empty
           if (max(video(currentrow,:,ii)) == 0)
               %if it's empty then just add to nolight
               nolight = nolight + 1;
           else
               nolight = 0;
               %get all positions of x and y
               for j = 1:cols
                  if (video(currentrow,j, ii) ~= 0) %if is bright
                     minx = min(minx, j);
                     maxx = max(maxx, j);
                     xsum = xsum + j;
                     ysum = ysum + currentrow;
                     pixelcount = pixelcount + 1;
                  end
               end
           end
           %next row
           currentrow = currentrow - 1;
       end
       x(ii) = round(xsum / pixelcount, 0);
       y(ii) = round(ysum / pixelcount, 0);
       drawbox(ymax,minx:maxx,ii) = 255;
       drawbox(currentrow,minx:maxx,ii) = 255;
       drawbox(currentrow:ymax,minx,ii) = 255;
       drawbox(currentrow:ymax, maxx, ii) = 255;
    end

    %flip y coordinate to switch from pixel to xy coordinate system
    y = -1 .* y;
end
```

Normalize.m

```
%normalizes all positions with mean of zero and standard deviation of one
function norm = Normalize(positions)
    for j = [2 4 6]
       [Y, MU, SIGMA] = zscore(positions(j,:));
```

```
        norm(j,:) = Y;
        MeanX = mean(positions(j-1,:));
        ZeroCenterX = positions(j-1,:) - MeanX;
        norm(j-1,:) = ZeroCenterX ./ SIGMA;
    end
```

PCAcode.m

```
%performs the SVD on positions
function [U, S, V, lambda, PCA, covariance] = PCAcode(positions)
    [m,n] = size(positions(1,:));
    [U,S,V] = svd(positions/sqrt(n-1),'econ');
    lambda = diag(S).^2;
    PCA = U' * positions;
    covariance = 1/(n-1) * positions * positions';
end
```