PCA

Andrei Karavanov

Tuesday, Feb. 26, 2019

Abstract

In this paper I will explore different scenarios when one could use principal component analysis to get an insight into the state of system.

Introduction and Overview

Imagine we perform a scientific experiment. In it we use three cameras, which are placed at the different angles, to record motion of the mass-spring system. Our main goal is to use this data to determine a governing equation of motion. Since the data from each camera records the same mass, in our case can, we would expect to see a lot of redundant information. In this paper we will discus four possible cases:

1. **Ideal Case**: Can moves only in z direction.
2. **Noisy Case**: Repeat ideal case, but introduce camera shake.
3. **Horizontal Displacement Case**: Can is released off-center so there will be motion both in the $x - y$ plane and z direction.
4. **Horizontal Displacement and Rotation Case**: Repeat horizontal displacement case, but introduce rotation.

Since we need at the max two sets of data as the output: z direction and the $x - y$ plane, we will approach this problem by using PCA. Specifically we will discuss whether low-dimensional reductions are appropriate to asses the dynamics of the mass-spring system.

Theoretical Background and Algorithm Implementation

From the introductory physics class about oscillations, we know that the governing equations for mass-spring system are:

$$\frac{d^2 f(t)}{dt^2} = -\omega^2 f(t) \tag{1}$$

If we will now solve this system, we will get the following solution:

$$f(t) = A\cos(\omega t + \omega_0) \tag{2}$$

which is a one-degree of freedom system. This solution is based on assumption that there is no movement in $x - y$ direction. This, however contradicts with case three and case four. Thus for cases three and four we don't know the governing equation prior the analysis of data. We however will assume that the result will be a two-degree of freedom system, since we would expect to see movement both in z direction and the $x - y$ plane.

Altogether, if we are correct with our assumptions and our governing equations are indeed one-degree of freedom and two-degrees of freedom systems, after applying PCA, we will get a low-dimensional systems, which will describe the motion of the system.

Now let us talk about our algorithmic approach.

First of all, let us talk about data ordering. In each case we will deal with three recorded footage. Each footage will consist of n number of frames. Each frame is represented by an RGB image in 2-D. That means our initial data set for each case is three 4-D matrices. Since it is not very practical to work with such structure we will use the following sudo-code to reorganize all of our data into a separate 2-D matrices:

```
Get the minimum amount of frames
For each video:
    For each frame:
        Move the frame from RGB to Grayscale
        Cast each value to a double
        Resize to the same size
        Reshape into array
        Store in data matrix
```

Now when we have a well structured matrices of each frame of each video, let us determine the location of the can at each frame:

```
For each video matrix:
    For each row:
        For each pixel:
            If pixel is in the central region and is brighter that 240:
                Store its coordinates in a matrix
    Find average brightest point for each frame and store its coordinates in vectors
Return vectors of X and Y coordinates of the average brightest point for each frame for
each video
```

Now, when we have the matrix of x and y coordinated (in pixels) of the can in each frame in each video, we organize that data in the following matrix:

$$\mathbf{X} = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \end{bmatrix} \tag{3}$$

Now when our data has been organized, let us tackle our main goals: redundancy and noise (case two). To asses, which data is redundant we will use the co-variance matrix:

$$\mathbf{C_X} = \frac{\mathbf{1}}{\mathbf{n-1}}\mathbf{XX^T} \tag{4}$$

where $\frac{1}{n-1}$ is our unbiased estimator. Each non-zero element located not on the diagonal shows that there exists redundancy in our data. Since, we want to eliminate this redundancy we want to diagonalize our co-variance matrix. Surprisingly enough, Singular Value Decomposition does exactly that. Here is how it works. Since our co-variance matrix is square matrix we can use eigenvalues decomposition to get the following:

$$\mathbf{XX^T} = \mathbf{Q\Sigma Q^{-1}} \tag{5}$$

Where Q is the matrix of eigenvectors and Σ is a diagonal matrix that represents distinct eigenvalues of C_X. Since our co-variance matrix it is symmetric, the eigenvector columns should be orthogonal. In other words, it follows that $Q^1 = Q^T$. That means we can rewrite our equation five as the following:

$$\mathbf{XX^T} = \mathbf{Q\Sigma Q^T} \tag{6}$$

These two facts suggest that we can try to work in the principal component basis.

$$\mathbf{Y} = \mathbf{Q^T X} \tag{7}$$

For this new basis, we can then consider its co-variance:

$$\begin{aligned} \mathbf{C_Y} &= \frac{\mathbf{1}}{\mathbf{n-1}}\mathbf{YY^T} = \frac{\mathbf{1}}{\mathbf{n-1}}(\mathbf{Q^T X})(\mathbf{Q^T X})^{\mathbf{T}} \\ &= \frac{\mathbf{1}}{\mathbf{n-1}}\mathbf{Q^T}(\mathbf{XX^T})\mathbf{Q} = \frac{\mathbf{1}}{\mathbf{n-1}}\mathbf{Q^T Q \Sigma Q^T Q} \\ &= \frac{\mathbf{1}}{\mathbf{n-1}}\mathbf{\Sigma} \end{aligned}$$

As we can see in principal component basis our co-variance matrix is diagonal. That means that all non-diagonal elements are zero, which means that we do not have any redundant information left! That is exactly what we wanted. The only thing left is to determine what information is important and what is not important. We can do that by looking at the singular values. The greater singular value, the greater "change" given principal component represent. In other words, if we will now find significantly important singular values and then will project our original data matrix on corresponding principal component vectors, we will end up with set of vectors that represent the projections of out data onto the dominant directions.

We will implement PCA using the following sudo-code:

De-mean our **X** matrix: subtract the mean of each row from each element of that row
Create co-variance matrix
Find eigenvalues and eigenvectors of our co-variance matrix
Sort our eigenvalues in decreasing order
Project our original data onto the principal component basis
Plot either one (cases one and two) or two (cases three and four) projection vectors that correspond to one/two largest singular values. singular values.

Computational Results

1. **Ideal Case**: Looking at figure one we can identify that first singular value dominates the rest by two order of magnitudes. That means that the most of motion in the mass-spring system can be represented by the projection of our original data onto that principal component. The rest of singular values tend to be less significant. In the ideal case we would expect to see projections on the rest principal components to be equal to zero, since we were are dealing with a one-degree of freedom system. In our case however, we can see that even though the rest of singular values are quite small they are not equal to zero. Same can be said about principal components. From the figure one we can see that even though second (blue), third (green) and fourth (yellow) principal components are not zero they do not provide any useful information about our system. The first principal component (red), in contrast, clearly shows that our system experiencing a simple harmonic oscillations.

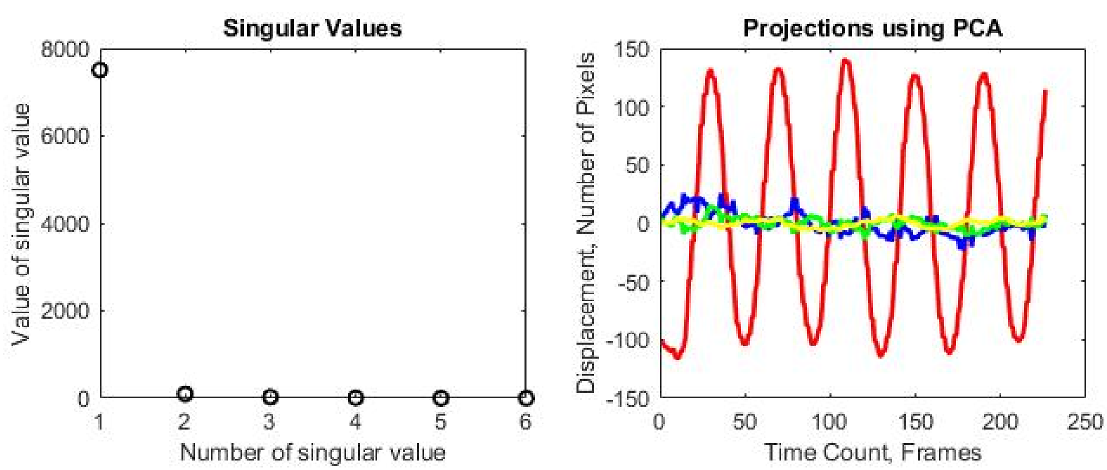

Figure 1: Results for the Ideal Case. Singular Values: [7503.5, 95.8, 24.9, 9.2, 0, 0]

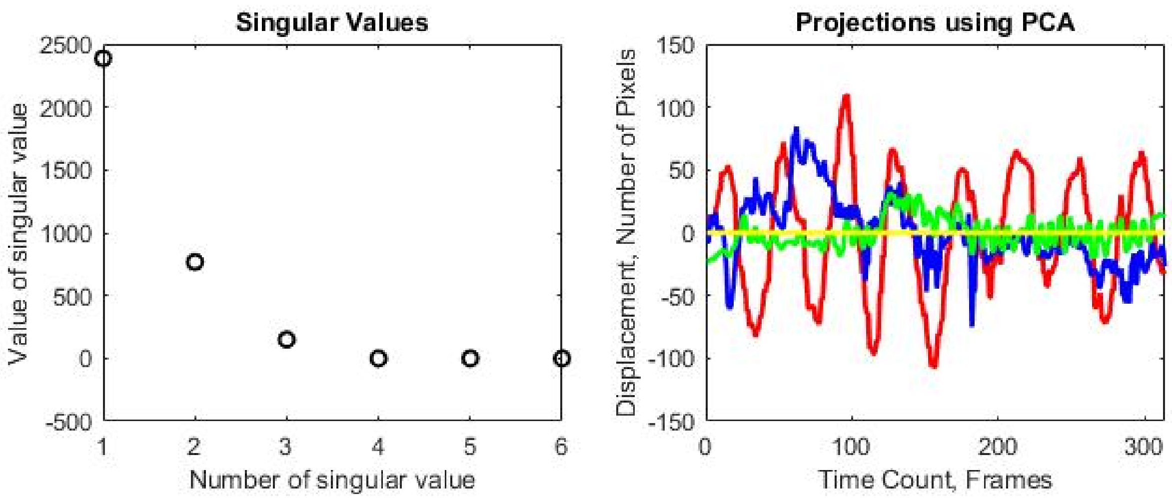

Figure 2: Results for the Noisy Case. Singular Values: [2386.6, 767.3, 150.5, 0, 0, 0]

2. **Noisy Case**: Unlike case one, case two introduces noise to our data. As we can see from the figure two, that leads to a different singular values as well as different projections. In this case we can see that the gap between the largest singular value and the second largest is not as drastic as it was in the case one. That comes from the fact that our algorithm had hard time finding redundant information in our initial data matrix. Nonetheless, if we will look at the vector projection on the first (red) principal component we will still see traits of harmonic motion. The rest of projections represent the noise that was coming from the shaking camera, and thus can be ignored.

3. **Horizontal Displacement Case**: In this case we release the can off-center, therefore we expect to see motion both in the $x-y$ plane and z direction. If we will look at the singular values plot on figure three we will see two singular values that clearly dominate the rest - first and second. It is important to notice that the greatest singular value is still five times greater than the second greatest. This behaviour can be explained by the fact that there was more motion in z direction rather motion in the $x-y$ plane. The rest singular values all stay around zero, which means that they are not significantly important. The same can be seen from the projection plot of figure three as well. We can clearly see how two projection on first (red) and second (blue) principal component look like oscillatory functions. The projections on third (green) and fourth (yellow) principal components, on the other hand, show no sign of oscillation and look closer to a constant line $y = 0$. Based on our observation of the projection behaviour, we indeed can conclude that given system is indeed a two-degree of freedom system.

4. **Horizontal Displacement and Rotation Case**: In this case we release the can off-center and give it some rotation. This case is similar to case three, except we will try to identify the rotational motion. From the singular values plot of figure four we can see that two singular values clearly dominate the rest - first and second. Similarly to previous case, we can notice that the greatest singular value is significantly greater than the second greatest. This behaviour, as well, can be explained by the fact that there

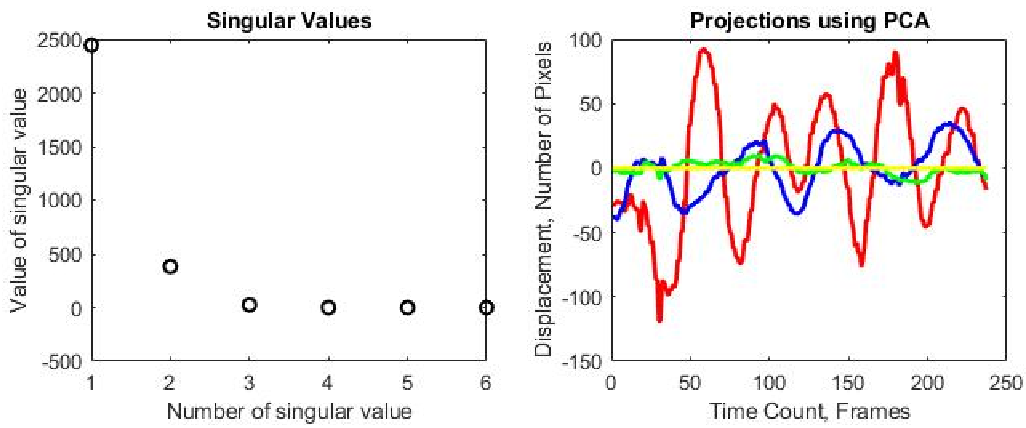

Figure 3: Results for the Horizontal Displacement Case. Singular Values: [2445.8, 383.0, 25.4, 0, 0, 0]

was more motion in z direction rather motion in the $x-y$ plane. The rest singular values all stay around zero, which means that they are not significantly important. Now, if we will look at the projection plot of figure four, we will clearly see how two projection on first (red) and second (blue) principal component look like oscillatory functions. Just like in the previous case, the projections on third (green) and fourth (yellow) principal components show no sign of oscillation and look closer to a constant line $y=0$. Altogether, we do not have any direct evidence that rotation was introduced into our system. The only one thing that have change are the periods of the oscillatory functions. In case four our model oscillates more violently. May be that is caused by rotation, may be no - we have no way of knowing that for sure based on our data. One thing that we can conclude is that given system is indeed a two-degree of freedom system.

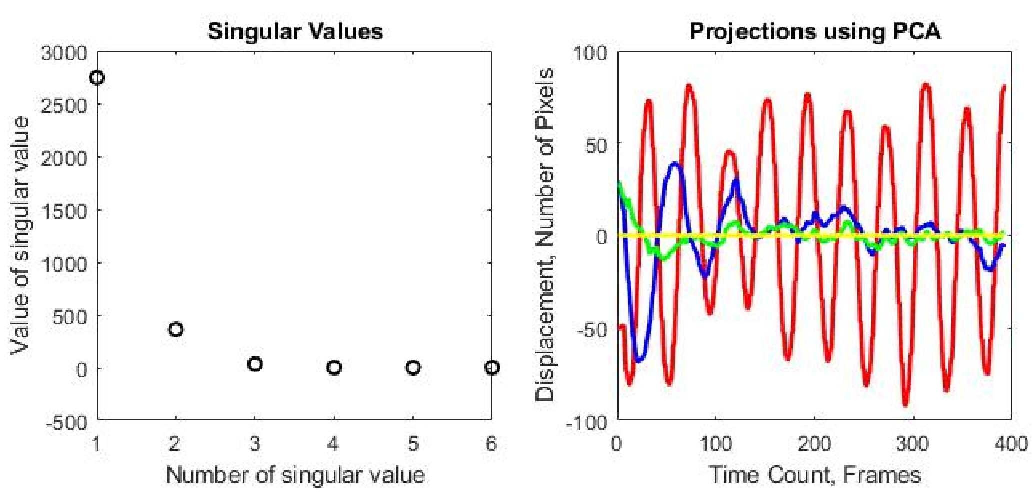

Figure 4: Results for the Horizontal Displacement and Rotation Case. Singular Values: [2744.5, 360.9, 33.7, 0, 0, 0]

Summary and Conclusions

In this paper we applied Principal Component Analysis to mass-spring system to determine a state of the system. We used four different scenarios to test whether PCA is the right tool or not. Here are our findings:

1. **Ideal Case**: In the ideal case the PCA performed extremely well. We were able to get a nice plot of the governing function of behaviour, which quite accurately represented the systems actual behaviour.

2. **Noisy Case**: The result of PCA was quite noisy. Although we were able to recognize the oscillatory behaviour in the system, we would not recommend using PCA on noisy data since the output will be noisy as well. If one have a noisy data and one wants to use PCA on it, one should de-noise it first if he/she wants to get a meaningful results out of PCA.

3. **Horizontal Displacement Case**: In the horizontal displacement case the PCA performed very well. It was able to identify both the movements in z direction and the $x - y$ plane.

4. **Horizontal Displacement and Rotation Case**: The result of PCA in this case left us with some question in our minds. For instance, we were not able to directly identify rotational movement of the system. Altogether, a bigger data set that will include 3-D mapping of objects will possibly deal with this uncertainty, however at the current state, usage of PCA on the given data-set is a 66-34 option. It successfully identifies movements in z direction and the $x - y$ plane, but misses the rotational movement.

To sum up all of the things, this paper a great insight into different scenarios when one could use principal component analysis to get an insight into the state of system. At the end we arrive to the conclusion whether the usage of PCA is effective or not for each specific scenario.

Appendices

A MATLAB commands

load(): Used to load video files into MatLab.
zeros(): Used to create empty matricies.
rgb2gray(): Used to move image from rgb space to grayscale space.
double(): Used to convert numbers to the double data type.
min(): Used to find the minimum element of the array.
size(): Used to find the size of the array.
imresize(): Used to resize images to the same dimensions.
reshape() : Used to reshape 2D matrix into 1D array.
rem(): Used to find reminder.
mean(): Used to find the mean of the data-set.
cov(): Used to find the co-variance matrix.
eig(): Used to find eigenvectors and eigenvalues.
sort(): Used to sort eigenvalues in decreasing order.
diag(): Used to find the diagonal of the matrix.

B MATLAB code

B.1 Main File

```
% HW3 - PCA
%% Test 1 - Initilization

clear all; close all; clc

disp("Getting Camera 1")
camOneRgb = load('cam1_1');
camOneRgb = camOneRgb.("vidFrames1_1");

disp("Getting Camera 2")
camTwoRgb = load('cam2_1');
camTwoRgb = camTwoRgb.("vidFrames2_1");

disp("Getting Camera 3")
camThreeRgb = load('cam3_1');
camThreeRgb = camThreeRgb.("vidFrames3_1");

% Test 1 - Setting-Up

close all; clc
```

```
timeSize = min([size(camOneRgb, 4) size(camTwoRgb, 4) size(camThreeRgb, 4)]);

xSize = min([size(camOneRgb, 1) size(camTwoRgb, 1) size(camThreeRgb, 1)]);

ySize = min([size(camOneRgb, 2) size(camTwoRgb, 2) size(camThreeRgb, 2)]);

camOneGrayMat = zeros(xSize, ySize, timeSize);
camTwoGrayMat = camOneGrayMat;
camThreeGrayMat = camOneGrayMat;

camOneGrayA = zeros(timeSize, xSize*ySize);
camTwoGrayA = camOneGrayA;
camThreeGrayA = camOneGrayA;

for i = 1:timeSize
    camOneGrayMat(:,:,i) = imresize(double(rgb2gray(camOneRgb(:,:,:,i))), ...
        [xSize,ySize]);
    camTwoGrayMat(:,:,i) = imresize(double(rgb2gray(camTwoRgb(:,:,:,i))), ...
        [xSize,ySize]);
    camThreeGrayMat(:,:,i) = imresize(double(rgb2gray(camThreeRgb(:,:,:,i))), ...
        [xSize,ySize]);
    camOneGrayA(i,:) = reshape(camOneGrayMat(:,:,i), [1, xSize*ySize]);
    camTwoGrayA(i,:) = reshape(camTwoGrayMat(:,:,i), [1, xSize*ySize]);
    camThreeGrayA(i,:) = reshape(camThreeGrayMat(:,:,i), [1, xSize*ySize]);
end

allCamsGrayA = [camOneGrayA; camTwoGrayA; camThreeGrayA];

allCamsGrayMat = zeros(xSize, ySize, timeSize, 3);
allCamsGrayMat(:,:,:,1) = camOneGrayMat;
allCamsGrayMat(:,:,:,2) = camTwoGrayMat;
allCamsGrayMat(:,:,:,3) = camThreeGrayMat;

leftBorders = 4*[70; 60; 60];
rightBorders = 4*[100; 90; 131];
```

```
[xCoordinates, yCoodinates] = findCoordinates(allCamsGrayMat,
    leftBorders, ...
    rightBorders, timeSize, xSize, ySize);

% Test 1 - PCA

%Creating X vector
xVec = zeros(6, timeSize);
for i = 1:6
    if rem(i, 2) == 0
        xVec(i, :) = yCoodinates(i/2, :);
    else
        xVec(i, :) = xCoordinates(rem(i,2),:);
    end
end

[m,n] = size(xVec);   % compute data size
mn = mean(xVec,2); % compute mean for each row
xVec = xVec - repmat(mn,1,n);  % subtract mean
covarianceMat = cov(xVec'); % compute covariance

[V,D]=eig(covarianceMat);       % eigenvectors(V)/eigenvalues(D)
lambda=diag(D);     % get eigenvalue

[dummy,m_arrange]=sort(-1*lambda);  % sort in decreasingorder
lambda=lambda(m_arrange);
V=V(:,m_arrange);

Y = V'*xVec;

% Test 1 - Output

subplot(1,2,1)
plot(lambda, 'ko', 'Linewidth', [1.5])
title('Singular Values')
xlabel('Number of singular value')
ylabel('Value of singular value')
subplot(1,2,2)
plot([1:timeSize], Y(1,:), "r", [1:timeSize], Y(2,:), "b", [1:
    timeSize], ...
    Y(3,:), "g", [1:timeSize], Y(4,:), "y", "Linewidth", [2])
title('Projections using PCA')
xlabel('Time Count, Frames')
ylabel('Displacement, Number of Pixels')
```

```
%% Test 2 - Initilization

clear all; close all; clc

disp("Getting Camera 1")
camOneRgb = load('cam1_2');
camOneRgb = camOneRgb.("vidFrames1_2");

disp("Getting Camera 2")
camTwoRgb = load('cam2_2');
camTwoRgb = camTwoRgb.("vidFrames2_2");

disp("Getting Camera 3")
camThreeRgb = load('cam3_2');
camThreeRgb = camThreeRgb.("vidFrames3_2");

% Test 2 - Setting-Up

close all; clc

timeSize = min([size(camOneRgb, 4) size(camTwoRgb, 4) size(camThreeRgb, 4)]);

xSize = min([size(camOneRgb, 1) size(camTwoRgb, 1) size(camThreeRgb, 1)]);

ySize = min([size(camOneRgb, 2) size(camTwoRgb, 2) size(camThreeRgb, 2)]);

camOneGrayMat = zeros(xSize, ySize, timeSize);
camTwoGrayMat = camOneGrayMat;
camThreeGrayMat = camOneGrayMat;

camOneGrayA = zeros(timeSize, xSize*ySize);
camTwoGrayA = camOneGrayA;
camThreeGrayA = camOneGrayA;

for i = 1:timeSize
    camOneGrayMat(:,:,i) = imresize(double(rgb2gray(camOneRgb(:,:,:,i))), ...
        [xSize,ySize]);
    camTwoGrayMat(:,:,i) = imresize(double(rgb2gray(camTwoRgb(:,:,:,i))), ...
        [xSize,ySize]);
```

```
    camThreeGrayMat(:,:,i) = imresize(double(rgb2gray(camThreeRgb(:,:,:,i))), ...
        [xSize,ySize]);
    camOneGrayA(i,:) = reshape(camOneGrayMat(:,:,i), [1, xSize*ySize]);
    camTwoGrayA(i,:) = reshape(camTwoGrayMat(:,:,i), [1, xSize*ySize]);
    camThreeGrayA(i,:) = reshape(camThreeGrayMat(:,:,i), [1, xSize*ySize]);
end

allCamsGrayA = [camOneGrayA; camTwoGrayA; camThreeGrayA];

allCamsGrayMat = zeros(xSize, ySize, timeSize, 3);
allCamsGrayMat(:,:,:,1) = camOneGrayMat;
allCamsGrayMat(:,:,:,2) = camTwoGrayMat;
allCamsGrayMat(:,:,:,3) = camThreeGrayMat;

leftBorders = 4*[70; 50; 60];
rightBorders = 4*[100; 100; 131];

[xCoordinates, yCoodinates] = findCoordinates(allCamsGrayMat, leftBorders, ...
    rightBorders, timeSize, xSize, ySize);

% Test 2 - PCA

%Creating X vector
xVec = zeros(6, timeSize);
for i = 1:5
    if rem(i, 2) == 0
        xVec(i, :) = yCoodinates(i/2, :);
    else
        xVec(i, :) = xCoordinates(rem(i,2),:);
    end
end

[m,n] = size(xVec);   % compute data size
mn = mean(xVec,2); % compute mean for each row
xVec = xVec - repmat(mn,1,n);  % subtract mean
covarianceMat = cov(xVec'); % compute covariance

[V,D]=eig(covarianceMat);      % eigenvectors(V)/eigenvalues(D)
lambda=diag(D);     % get eigenvalue

```

```
[dummy, m_arrange]=sort(-1*lambda);  % sort in decreasingorder
lambda=lambda(m_arrange);
V=V(:,m_arrange);

Y = V'*xVec;

% Test 2 - Output

subplot(1,2,1)
plot(lambda, 'ko', 'Linewidth', [1.5])
title('Singular Values')
xlabel('Number of singular value')
ylabel('Value of singular value')
subplot(1,2,2)
plot([1:timeSize], Y(1,:), "r", [1:timeSize], Y(2,:), "b", [1:timeSize], ...
    Y(3,:), "g", [1:timeSize], Y(4,:), "y", "Linewidth", [2])
title('Projections using PCA')
xlabel('Time Count, Frames')
ylabel('Displacement, Number of Pixels')

%% Test 3 - Initilization

clear all; close all; clc

disp("Getting Camera 1")
camOneRgb = load('cam1_3');
camOneRgb = camOneRgb.("vidFrames1_3");

disp("Getting Camera 2")
camTwoRgb = load('cam2_3');
camTwoRgb = camTwoRgb.("vidFrames2_3");

disp("Getting Camera 3")
camThreeRgb = load('cam3_3');
camThreeRgb = camThreeRgb.("vidFrames3_3");

% Test 3 - Setting-Up

close all; clc

timeSize = min([size(camOneRgb, 4) size(camTwoRgb, 4) size(camThreeRgb, 4)]);
```

```
xSize = min([size(camOneRgb, 1) size(camTwoRgb, 1) size(
    camThreeRgb, 1)]);

ySize = min([size(camOneRgb, 2) size(camTwoRgb, 2) size(
    camThreeRgb, 2)]);

camOneGrayMat = zeros(xSize, ySize, timeSize);
camTwoGrayMat = camOneGrayMat;
camThreeGrayMat = camOneGrayMat;

camOneGrayA = zeros(timeSize, xSize*ySize);
camTwoGrayA = camOneGrayA;
camThreeGrayA = camOneGrayA;

for i = 1:timeSize
    camOneGrayMat(:,:,i) = imresize(double(rgb2gray(camOneRgb
        (:,:,:,i))), ...
         [xSize,ySize]);
    camTwoGrayMat(:,:,i) = imresize(double(rgb2gray(camTwoRgb
        (:,:,:,i))), ...
         [xSize,ySize]);
    camThreeGrayMat(:,:,i) = imresize(double(rgb2gray(camThreeRgb
        (:,:,:,i))), ...
         [xSize,ySize]);
    camOneGrayA(i,:) = reshape(camOneGrayMat(:,:,i), [1, xSize*
        ySize]);
    camTwoGrayA(i,:) = reshape(camTwoGrayMat(:,:,i), [1, xSize*
        ySize]);
    camThreeGrayA(i,:) = reshape(camThreeGrayMat(:,:,i), [1, xSize
        *ySize]);
end

allCamsGrayA = [camOneGrayA; camTwoGrayA; camThreeGrayA];

allCamsGrayMat = zeros(xSize, ySize, timeSize, 3);
allCamsGrayMat(:,:,:,1) = camOneGrayMat;
allCamsGrayMat(:,:,:,2) = camTwoGrayMat;
allCamsGrayMat(:,:,:,3) = camThreeGrayMat;

leftBorders = 4*[70; 60; 60];
rightBorders = 4*[100; 90; 131];

[xCoordinates, yCoodinates] = findCoordinates(allCamsGrayMat,
    leftBorders, ...
     rightBorders, timeSize, xSize, ySize);
```

```
% Test 3 - PCA

%Creating X vector
xVec = zeros(6, timeSize);
for i = 1:5
    if rem(i, 2) == 0
        xVec(i, :) = yCoodinates(i/2, :);
    else
        xVec(i, :) = xCoordinates(rem(i,2),:);
    end
end

[m,n] = size(xVec);   % compute data size
mn = mean(xVec,2); % compute mean for each row
xVec = xVec - repmat(mn,1,n);  % subtract mean
covarianceMat = cov(xVec'); % compute covariance

[V,D]=eig(covarianceMat);        % eigenvectors(V)/eigenvalues(D)
lambda=diag(D);     % get eigenvalue

[dummy,m_arrange]=sort(-1*lambda);  % sort in decreasingorder
lambda=lambda(m_arrange);
V=V(:,m_arrange);

Y = V'*xVec;

% Test 3 - Output

subplot(1,2,1)
plot(lambda, 'ko', 'Linewidth', [1.5])
title('Singular Values')
xlabel('Number of singular value')
ylabel('Value of singular value')
subplot(1,2,2)
plot([1:timeSize], Y(1,:), "r", [1:timeSize], Y(2,:), "b", [1:
    timeSize], ...
    Y(3,:), "g", [1:timeSize], Y(4,:), "y", "Linewidth", [2])
title('Projections using PCA')
xlabel('Time Count, Frames')
ylabel('Displacement, Number of Pixels')

%% Test 4 - Initilization

clear all; close all; clc
```

```
disp("Getting Camera 1")
camOneRgb = load('cam1_4');
camOneRgb = camOneRgb.("vidFrames1_4");

disp("Getting Camera 2")
camTwoRgb = load('cam2_4');
camTwoRgb = camTwoRgb.("vidFrames2_4");

disp("Getting Camera 3")
camThreeRgb = load('cam3_4');
camThreeRgb = camThreeRgb.("vidFrames3_4");

% Test 4 - Setting-Up

close all; clc

timeSize = min([size(camOneRgb, 4) size(camTwoRgb, 4) size(
    camThreeRgb, 4)]);

xSize = min([size(camOneRgb, 1) size(camTwoRgb, 1) size(
    camThreeRgb, 1)]);

ySize = min([size(camOneRgb, 2) size(camTwoRgb, 2) size(
    camThreeRgb, 2)]);

camOneGrayMat = zeros(xSize, ySize, timeSize);
camTwoGrayMat = camOneGrayMat;
camThreeGrayMat = camOneGrayMat;

camOneGrayA = zeros(timeSize, xSize*ySize);
camTwoGrayA = camOneGrayA;
camThreeGrayA = camOneGrayA;

for i = 1:timeSize
    camOneGrayMat(:,:,i) = imresize(double(rgb2gray(camOneRgb
        (:,:,:,i))), ...
         [xSize,ySize]);
    camTwoGrayMat(:,:,i) = imresize(double(rgb2gray(camTwoRgb
        (:,:,:,i))), ...
         [xSize,ySize]);
    camThreeGrayMat(:,:,i) = imresize(double(rgb2gray(camThreeRgb
        (:,:,:,i))), ...
         [xSize,ySize]);
```

```
    camOneGrayA(i,:) = reshape(camOneGrayMat(:,:,i), [1, xSize*
        ySize]);
    camTwoGrayA(i,:) = reshape(camTwoGrayMat(:,:,i), [1, xSize*
        ySize]);
    camThreeGrayA(i,:) = reshape(camThreeGrayMat(:,:,i), [1, xSize
        *ySize]);
end

allCamsGrayA = [camOneGrayA; camTwoGrayA; camThreeGrayA];

allCamsGrayMat = zeros(xSize, ySize, timeSize, 3);
allCamsGrayMat(:,:,:,1) = camOneGrayMat;
allCamsGrayMat(:,:,:,2) = camTwoGrayMat;
allCamsGrayMat(:,:,:,3) = camThreeGrayMat;

leftBorders = 4*[70; 60; 60];
rightBorders = 4*[100; 90; 131];

[xCoordinates, yCoodinates] = findCoordinates(allCamsGrayMat,
    leftBorders, ...
    rightBorders, timeSize, xSize, ySize);

% Test 4 - PCA

%Creating X vector
xVec = zeros(6, timeSize);
for i = 1:5
    if rem(i, 2) == 0
        xVec(i, :) = yCoodinates(i/2, :);
    else
        xVec(i, :) = xCoordinates(rem(i,2),:);
    end
end

[m,n] = size(xVec);   % compute data size
mn = mean(xVec,2); % compute mean for each row
xVec = xVec - repmat(mn,1,n);  % subtract mean
covarianceMat = cov(xVec'); % compute covariance

[V,D]=eig(covarianceMat);      % eigenvectors(V)/eigenvalues(D)
lambda=diag(D);    % get eigenvalue

[dummy,m_arrange]=sort(-1*lambda);  % sort in decreasingorder
lambda=lambda(m_arrange);
V=V(:,m_arrange);
```

```
Y = V'*xVec;

% Test 4 - Output

subplot(1,2,1)
plot(lambda, 'ko', 'Linewidth', [1.5])
title('Singular Values')
xlabel('Number of singular value')
ylabel('Value of singular value')
subplot(1,2,2)
plot([1:timeSize], Y(1,:), "r", [1:timeSize], Y(2,:), "b", [1:timeSize], ...
    Y(3,:), "g", [1:timeSize], Y(4,:), "y", "Linewidth", [2])
title('Projections using PCA')
xlabel('Time Count, Frames')
ylabel('Displacement, Number of Pixels')
```

B.2 Script that finds coordinates

```
function [xCoordinates, yCoodinates] = findCoordinates(data, leftBorder, ...
    rightBorder, timeSize, xSize, ySize)

xCoordinates = zeros(3,timeSize);
yCoodinates = zeros(3,timeSize);
counts = zeros(3,timeSize);

for i = 1:timeSize
    for j = 1:xSize
        for k = 1:ySize
            for l = 1:3
                if data(j,k,i,l) >= 240 && k >= leftBorder(l) && k <= rightBorder(l)
                xCoordinates(l, i) = xCoordinates(l, i) + j;
                yCoodinates(l, i) = yCoodinates(l, i) + k;
                counts(l, i) = counts(l, i) + 1;
                end
            end
        end
    end

    for j = 1:3
        xCoordinates(j, i) = xCoordinates(j, i)/counts(j, i);
        yCoodinates(j, i) = yCoodinates(j, i)/counts(j, i);
```

```
        end
end
end
```