Homework 3: Primary Component Analysis

Maximilian Walter
AMATH 482
University of Washington

This assignment uses principal component analysis (PCA) to analyze data from three cameras. Each camera is recording the oscillatory motion of a paint can on a spring from a different angle. We are given four different scenarios, an ideal case and noisy case of a single dimensional motion, then two cases were the can is rotating while undergoing a pendulum-like potion. Using PCA we can reduce the dimensions to evaluate the motion of the can by looking at the energy content of each component.

Introduction and Overview

We start this experiment with four sets of data provided by three different cameras. Each camera is at a different angle and is recording the motion of a paint can with a flashlight on top moving on a spring. In each trial, the can is undergoing a different type of motion, ideal up/down motion, a noisy version of this data where the camera is shaken during the trial, a version where the can is swung like a pendulum while also oscillating vertically, and the final trial where the can is undergoing all of the previous motions as well as rotation. Although this information is provided, we are told to do the assignment without this knowledge.

Once the data is aligned, it can be decomposed into its singular values using the SVD command. Looking at the singular values, the energy from each component can be extracted to understand how much information is stored. Using this information, the signal can be recreated to track the can in lower dimensionality. The matrix can be reduced from 6 x Frames to the 1-3 x Frames depending on the energy captured. With this information, it would be possible to plot the motion of the can without seeing the video.

Theoretical Background

SVD

Singular Value Decomposition is a process of manipulating a matrix, A, into three components [U, S, V]. Each component contains information about the original matrix. V contains the shape of our data, S then stretches or shrinks the data, and U rotates it to a new axis. The idea behind this is to collapse the data's dimensions and shows the variance within the dataset. The process of computing SVD is as follows

$$A = U\Sigma V^* \tag{1}$$

$$A^T A = (U\Sigma V^*)^T (U\Sigma V^*) \; note\; V^*\; is\; V\; transposed$$

$$A^T A = V\Sigma U^* (U\Sigma V^*)$$

$$A^T A = V\Sigma^2 V^* \tag{1.1}$$

And

$$A^T A = (U\Sigma \mathrm{V}^*)(U\Sigma \mathrm{V}^*)^T$$
$$A^T A = U\Sigma \mathrm{V}^*(V\Sigma \mathrm{U}^*)$$
$$A^T A = U\Sigma^2 U^* \quad 1.2$$

Multiplying 1.1 and 1.2 by U and V gives two self-consistent eigenvalue problems.

$$A^T AV = V\Sigma^2 \quad 1.3$$
$$AA^T U = V\mathrm{U} \quad 1.4$$

Solving these equations for the normalized eigenvectors, we can find the eigenvalues are the square of the singular values themselves or in reverse, the square root of the eigenvalues produces the singular values.

PCA

Using SVD, we can diagonalize our data, thus finding the covariance of the matrix via the off-diagonal values. SVD also conveniently places information from A within two new bases, this information is stored in U and V. Evaluating the singular values, the energy or amount of data captured within this value, can be used to determine what components should be kept in the new basis. After evaluating each component's energy, a new matrix can be formed by multiplying on the components needed by the original matrix.

$$Y = U^*(1)A \quad 2$$

This reduces the number of dimensions needed to explain your data from a higher order to the number of primary components needed to represent a majority of your data.

Algorithm Development

The data from each camera was loaded and the number of frames was extracted. From this information the length of each video was different, so the first task was to align the data.

Tracking

Each video was cropped to only include the motion of the can. To track the can in space, the image was converted to grayscale. Because of the flashlight attached to the can, a threshold can be set to only track the brightest points in the image (> 250-pixel value). Then the matrix index is extracting using the find command, to recreate a new matrix, with the X and Y motion for each frame. The mean value is taken and stored for each frame. Because of how matrices are indexed, the Y value must be flipped by subtracting the mean vector from 480 (the size of the video matrix). This process is done for each camera. The mean Y vector was plotted for camera 1 and 2, and for camera 3, the mean X values were used. This is because these directions included the "most" motion information for preliminary plots.

Alignment

After the motion is extracted from each camera, the first minimum is found to align the data. Using this information, each X/Y vector was fit to the minimum value index. The overhanging data was cut to the shortest recording. Each vector was then concatenated into one

large matrix with dimension 6 x Frames (X, Y for each camera). The mean was taken of each row and subtracted from the data to center the data. This was done by finding the mean of each value then repopulating a matrix, that repeats this column vector to the size of our full data set.

SVD

Once all the data is aligned and, in a matrix, the *SVD* command decomposes the matrix into U, S, V. The eigenvalues were extracted by taking the singular values, which is the diagonal of our S matrix, squared and stored as lambda. The energy can then be extracted by normalizing lambda over the sum of lambda. The singular values can be used to create a projection of A, by using the principle components that contain the most data. This entire process was done for all four trials.

Computational Results

Trial #1

For the first trial, the "ideal" case, the principal components showed that almost all the data, 98%, was captured in the rank 1 approximation. This makes sense as the can was moving mostly in one direction. The results can be seen in Figure 1.

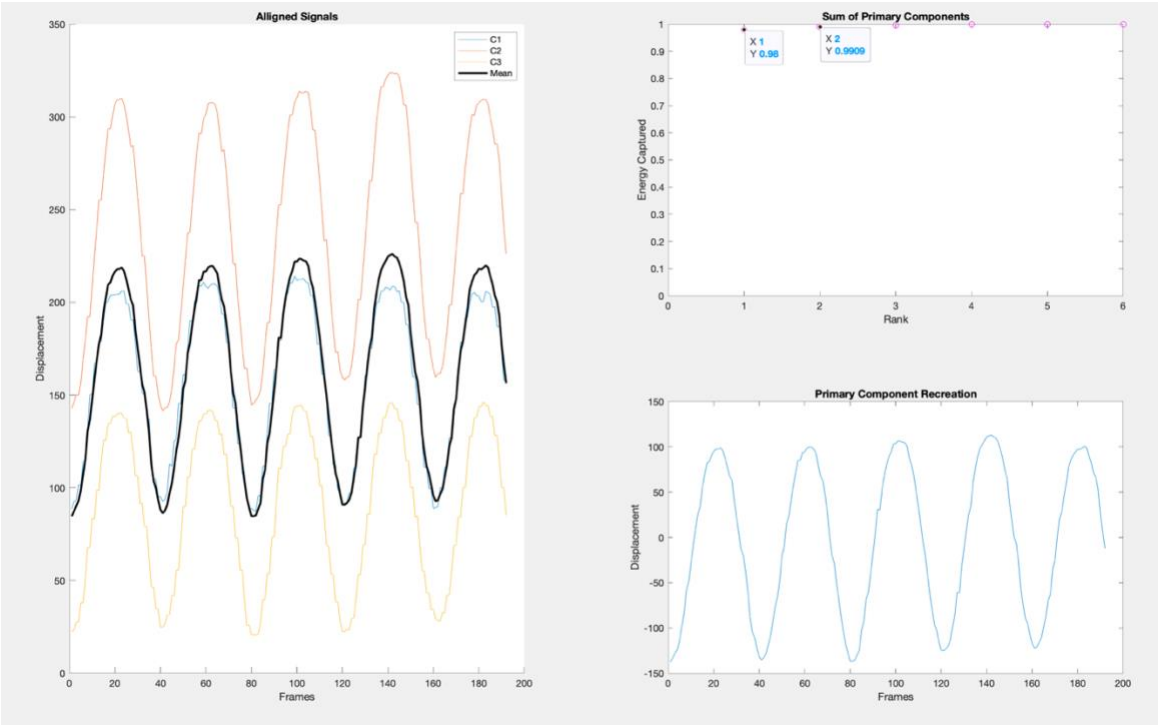

Figure 1. The left plot shows the aligned motion of the can with the mean value overlaid. The top right plot shows the sum of primary components, with PC 1 capturing 98% of the data. The bottom right plot is the recreation of motion using PC 1.

Trial #2

In this case, the primary component analysis resulted in the first component contains 66% of the data. Using a rank 3 approximation, we achieve a 94.5% capturing of the cans motion.

Even though only 66% is shown in PC 1, the projection still captures most of the shape, as seen in figure 2, but loses the specificity in the peaks. The decline in energy captured is mostly due to a large amount of noise in the original recordings. The noise adds information in all directions (X, Y, Z), and thus most of the data can be captured with a rank 3 approximation but would be confusing without the original video available. The overall oscillations are still captured.

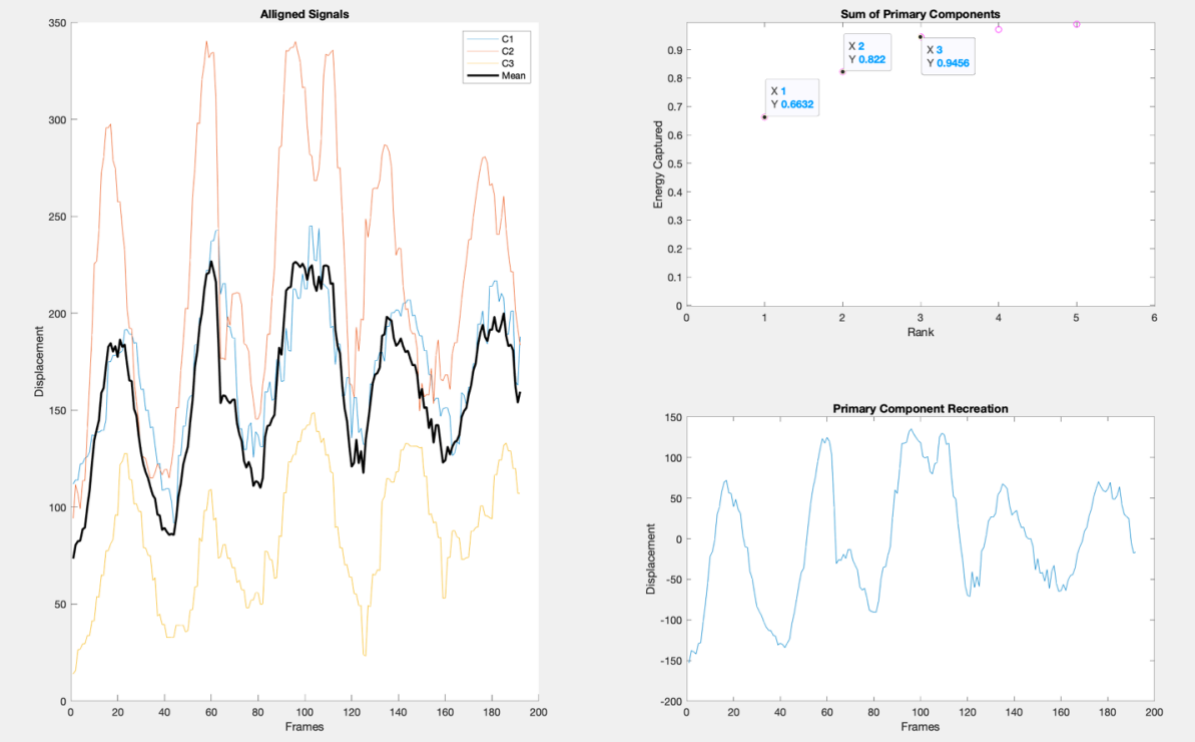

Figure 2. The left plot shows the aligned motion of the can with the mean value overlaid. The top right plot shows the sum of primary components, with PC 1 capturing 66% of the data. The bottom right plot is the recreation of motion using PC 1.

Trial #3

In this trial, the energy captured in the first principle component is 68.77% and with a rank 2 approximation, 98% of the data is captured. This leads me to believe that there is a multi-directional movement. Almost all the data is captured in the rank 2 approximation. This is a bit odd as there are three components at play, the X, Y plane as well as the Z plane.

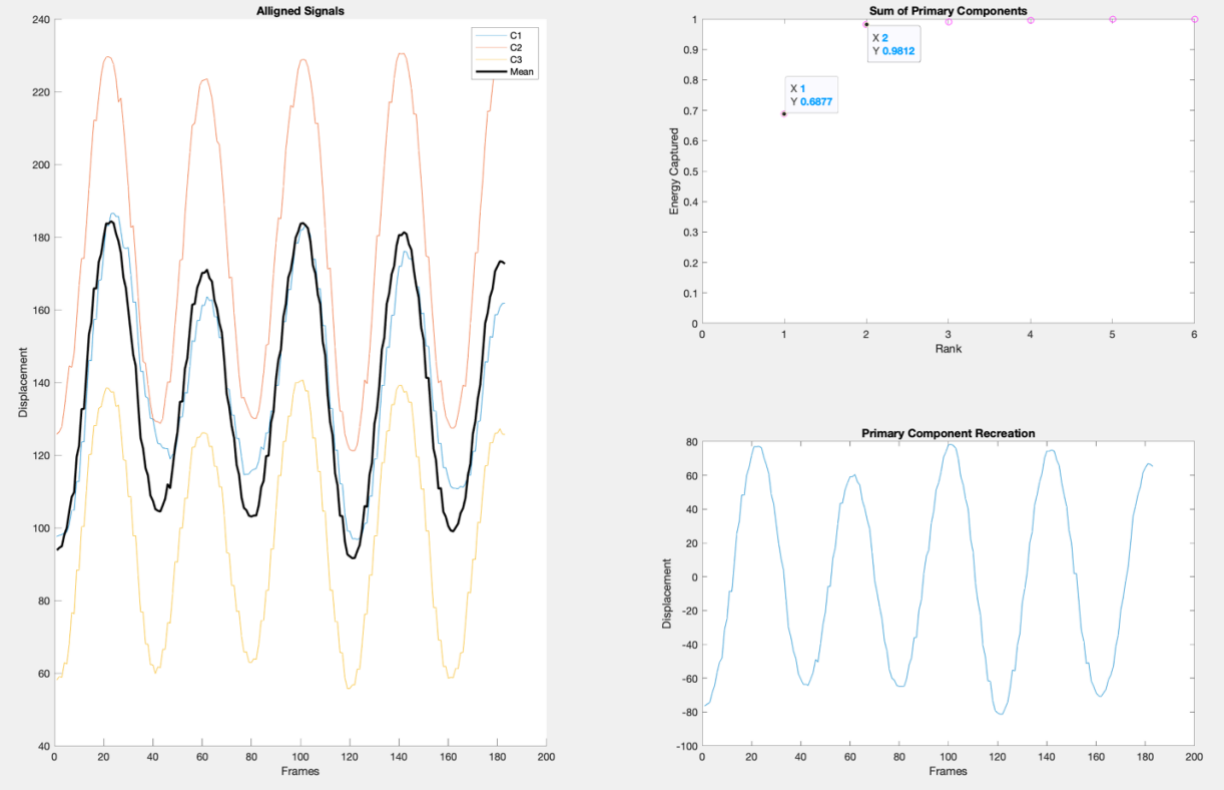

Figure 3. The left plot shows the aligned motion of the can with the mean value overlaid. The top right plot shows the sum of primary components, with PC 1 capturing 68.8% of the data. The bottom right plot is the recreation of motion using PC 1.

Trial #4

In this trial, PC 1 contains 76.8% of the data, with the rank 3 approximation containing 96& of the data. This shows at least three dimensions of movement, in a clear oscillation pattern. Unlike trial 3, we do capture the third motion. It doesn't clearly show the rotation aspect the can but does hint to a different type of motion.

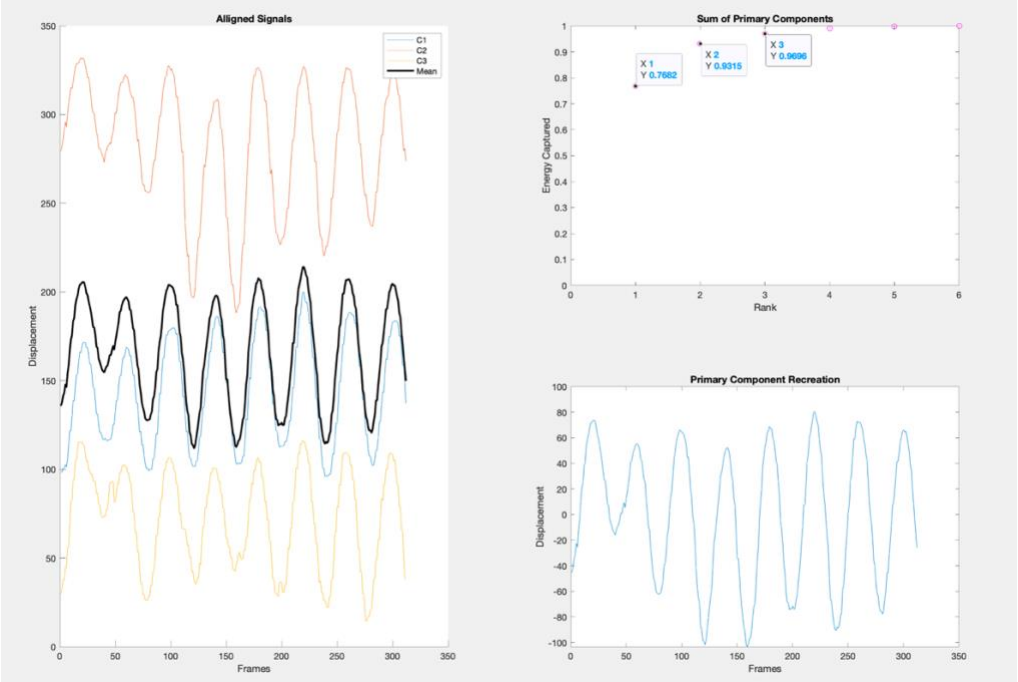

Figure 4. The left plot shows the aligned motion of the can with the mean value overlaid. The top right plot shows the sum of primary components, with PC 1 capturing 76.8% of the data. The bottom right plot is the recreation of motion using PC 1.

Summary and Conclusion

From this experiment, we were able to break down a large data set from video and discover the processes occurring in that video. In each trial, the PCA was able to capture the overall oscillations of the paint can. Trial 3-4 did hint that there was a bit more going on in the video compared to the ideal cases but does not help distinguish directly rotation or horizontal movement.

Appendix A

- *Rgb2gray(X)*: This function converts the pixels in an image into a grayscale.
- *Uint8(X)*: Converts a vector or matrix into a uint8 data type.
- *Ind2sub(*size, index): Turns indices into a matrix of the appropriate size. This is used to fill in the points of interest when parsing through the grayscale image.
- *Find(X):* Returns a vector of non-zero indices in a matrix. Used to find where color values correspond to being above a threshold.
- *Min(X)*: Finds the minimum value. This function is used to align the data to the first minimum of each plot.
- *Mean(X,2)*: This finds the mean value across the second dimension of a matrix (Rows).
- *Diag(S):* Returns the values on the diagonal of a matrix. Used to extract singular values from our S matrix after taking the SVD.

Appendix B:
Code from only one trial is included as it was repeated for every other trial.

```
%AMATH 482
%MAX WALTER
%HOMEWORK 3
clear all; close all; clc

%Loading all data
load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')
numFrames1 = size(vidFrames1_1,4);
numFrames2 = size(vidFrames2_1,4);
numFrames3 = size(vidFrames3_1,4);

%%
%Data 1

%Creating Frame for bucket
frame = zeros(480,640);
frame(200:430,300:400) = 1;
frame_u8 = uint8(frame);


%Initializing In-Loop Values
mean_x1 = zeros(1,length(numFrames1));
mean_y1 = zeros(1,length(numFrames1));


for j = 1:numFrames1

    %Process Video so that it's in grayscale and framed for just the bucket.
    X1 = vidFrames1_1(:,:,:,j); %Loading Image Frames
    X1_g = rgb2gray(X1); %Converting to Gray
    fX1_g = X1_g .* frame_u8; %Frame only the bucket moving.
```

```
    %Focus on the Light by making B/W
    light1 = fX1_g > 250;
    idx1 = find(light1);

    %Returning data into Matrix form.
    [Y,X]  = ind2sub(size(light1),idx1);

    %Since there are two points, finding the mean point for the can
    mean_x1(j) = mean(X);
    mean_y1(j) = mean(Y);
end

%Inverting image
corrected_image1 = 480 - mean_y1;

%Plotting
figure()
hold on
plot(1:numFrames1,corrected_image1)

%%
%Data 2

%Creating Frame for bucket
frame = zeros(480,640);
frame(100:385,260:340) = 1;
frame_u8 = uint8(frame);

%Initializing In-Loop Values
mean_x2 = zeros(1,length(numFrames2));
mean_y2 = zeros(1,length(numFrames2));


for j = 1:numFrames2

    %Process Video so that it's in grayscale and framed for just the bucket.
    X2 = vidFrames2_1(:,:,:,j); %Loading Image Frames
    X2_g = rgb2gray(X2); %Converting to Gray
    fX2_g = X2_g .* frame_u8; %Frame only the bucket moving.


    %Focus on the Light by making B/W
    light2 = fX2_g > 240;
    idx2 = find(light2);

    %Returning data into Matrix form.
    [Y,X]  = ind2sub(size(light2),idx2);

    %Since there are two points, finding the mean point for the can
    mean_x2(j) = mean(X);
    mean_y2(j) = mean(Y);
end
```

```
%Inverting image
corrected_image2 = 480 - mean_y2;

%Plotting
plot(1:numFrames2,corrected_image2)

%%
%Data 3

%Creating Frame for bucket
frame = zeros(480,640);
frame(190:330,270:485) = 1;
frame_u8 = uint8(frame);

%Initializing In-Loop Values
mean_x3 = zeros(1,length(numFrames3));
mean_y3 = zeros(1,length(numFrames3));


for j = 1:numFrames3

    %Process Video so that it's in grayscale and framed for just the bucket.
    X3 = vidFrames3_1(:,:,:,j); %Loading Image Frames
    X3_g = rgb2gray(X3); %Converting to Gray
    fX3_g = X3_g .* frame_u8; %Frame only the bucket moving.

    %Focus on the Light by making B/W
    light3 = fX3_g > 235;
    idx3 = find(light3);

    %Returning data into Matrix form.
    [Y,X]  = ind2sub(size(light3),idx3);

    %Since there are two points, finding the mean point for the can
    mean_x3(j) = mean(X);
    mean_y3(j) = mean(Y);
end

%Inverting image, using X component as the video is titled
corrected_image3 = 480  - mean_x3;

%Plotting
plot(1:numFrames3,corrected_image3)
title('Uncentered Plots')
xlabel('Frames')
ylabel('Displacement')
legend('C1','C2','C3')

%%
%All data

%Alligning Data
[M,I1] = min(corrected_image1(1,[1:75]));
min1 = I1;
```

```
[M,I2] = min(corrected_image2(1,[1:75]));
min2 = I2;

[M,I3] = min(corrected_image3(1,[1:75]));
min3 = I3;



width = 191;
shift_img1 = 480 - mean_y1(min1:min1+width);
shift_img2 = 480 - mean_y2(min2:min2+width);
shift_img3 = 480 - mean_x3(min3:min3+width);

Avg_M = [shift_img1; shift_img2; shift_img3];
mean_M = mean(Avg_M);


figure()
subplot(2,2,[1 3])
hold on
plot(1:width+1,shift_img1)
plot(1:width+1,shift_img2)
plot(1:width+1,shift_img3)
plot(1:width+1,mean_M,'k','Linewidth',2)
title('Alligned Signals')
xlabel('Frames')
ylabel('Displacement')
legend('C1','C2','C3','Mean')

Full_set = [mean_x1(min1:min1+width); shift_img1; mean_x2(min2:min2+width);
shift_img2; shift_img3; mean_y3(min3:min3+width);];
[m,n] = size(Full_set);
mean_FS = mean(Full_set,2);
Full_set = Full_set - repmat(mean_FS,1,n);


[U,S,V] = svd(Full_set/sqrt(n-1));

lambda = diag(S).^2;
sigma = diag(S);
energy = lambda/sum(lambda);

proj = U'*Full_set;

subplot(2,2,2)
plot(1:6, [energy(1) sum(energy(1:2)) sum(energy(1:3)) sum(energy(1:4))
sum(energy(1:5)) sum(energy(1:6))],'mo')
ylim([0 1]);
xlim([0 6]);
title('Sum of Primary Components')
xlabel('Rank')
ylabel('Energy Captured')
```

```
subplot(2,2,4)
plot(1:192,proj(1,:))
title('Primary Component Recreation')
xlabel('Frames')
ylabel('Displacement')
```