AMATH 482 Homework 3

Zach Zlepper

Feb 24, 2020

Abstract

In this paper, we will use PCA to pull out core components from the motion of a paint can. We are given three camera angles, four different cases, and n frames. In our calculation, we will build a large matrix which describes the motion of the can. This matrix will then be sent through the PCA algorithm to find the important features of this system by looking at the energies of each component.

1 Introduction and Overview

Sometimes it is hard to understand the underlying components of a data set. When we are given data with high correlation and overlap, we must use algorithms to pull out uncorrelated features of the system. Throughout this paper, we will investigate a spring mass system from 3 camera angles. The extra camera angles gives us correlated data while giving us a little new information. By oversampling, we can pick up on complex behaviors such as rotation, swinging, and oscillating. We will also show that we can see through noise with the SVD algorithm.

1.1 Problem Overview

In this assignment we are given four cases to analyze with each having three different cameras filming. Each case is unique and teaches us a trait about the PCA. The cases are as follows:

1. Ideal Case: Vertical motion
2. Noisy Case: Vertical motion with noise
3. Horizontal Displacement: Vertical motion with horizontal movement
4. Horizontal Displacement and Rotation: spinning the can with vertical and horizontal motion

2 Theoretical Background

In order to get to Principle component analysis, we must first discuss the Singular Value Decomposition. The SVD is a Decomposition of a matrix A such that

$$\begin{gathered}\mathbf{A} = \mathbf{U\Sigma V}^* \text{ where} \\ \mathbf{U} \in \mathbb{C}^{m\times m} \text{ is unitary} \\ \mathbf{V} \in \mathbb{C}^{n\times n} \text{ is unitary} \\ \mathbf{\Sigma} \in \mathbb{R}^{m\times n} \text{ is diagonal}\end{gathered}$$

The SVD has a geometric interpretation as well which is as follows:

- Multiplying by V^* rotates our vectors
- Multiplying by Σ stretches our vectors
- Multiplying by U rotates our vectors again

So how do we compute such a decomposition? The answers is we find the eigenvalues and eigenvectors of the matrix $A^T A$ and AA^T

$$\begin{aligned}
\mathbf{A}^T\mathbf{A} &= (\mathbf{U\Sigma V^*})^T (\mathbf{U\Sigma V^*}) \\
&= \mathbf{V\Sigma U^* U\Sigma V^*} \\
&= \mathbf{V\Sigma^2 V^*}
\end{aligned}$$

$$\begin{aligned}
\mathbf{AA}^T &= (\mathbf{U\Sigma V^*}) (\mathbf{U\Sigma V^*})^T \\
&= \mathbf{U\Sigma V^* V\Sigma U^*} \\
&= \mathbf{U\Sigma^2 U^*}
\end{aligned}$$

$$\begin{aligned}
\mathbf{A}^T\mathbf{AV} &= \mathbf{V}\Sigma^2 \\
\mathbf{AA}^T\mathbf{U} &= \mathbf{U\Sigma}^2
\end{aligned}$$

We can see the U and V matrices contain the eigenvectors of AA^T and $A^T A$ respectively with Σ^2 as the eigenvalues of both AA^T and $A^T A$. The square root of the eigenvalues of these equations produces the singular values which are an output in the SVD.

We will use the SVD as the backbone algorithm for the PCA which allows high dimensional data to be deconstructed to lower dimensional representations. In addition, the larger the singular values the more energy. This fact when used in PCA allows us to see which principle components are more present in the system. We will add some statistical theory such as the the covariance matrix to display the property that the SVD have an uncorrelated components. Below we show that the components that the SVD yields are uncorrelated. $\mathbf{C_X} = \frac{1}{n-1}\mathbf{XX}^T$, where C_X is a square $m \times m$. We define the transformed variable $Y = U * X$ where U is the unitary transformation associated with the SVD ($\mathbf{X} = \mathbf{U\Sigma V^*}$). We then calculate the variance in Y:

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

This calculation shows that the covariance matrix of SVD components yields an uncorrelated decomposition of the data becasue the off diagnal values are 0. Not only are they uncorrelated but they also can be broken down to find the most important component to the signal by using the singular values.

3 Algorithm Implementation and Development

In order to calculate the principle components of the motion, we must first process the video and build a matrix with our data as displayed in Algorithm 1. We must then repeat Algorithm 1 for every camera angle we have. We must combined the matrices produced in Algorithm 1 to build a large matrix with each measurement, X or Y for each camera, in its own row and each observation, frames, in its own column. This is encapsulated in Algorithm 2 along with doing the PCA.

Algorithm 1: Light Tracking

```
Initialize video
for Every Frame do
    Convert image to a double in gray scale
    Find location of all white values about certain threshold
    Find the average x and y index of "white" box
    Store mean values in a two column matrix
end for
```

Algorithm 2: PCA

```
Cut light tracking data to the same length for all cameras
Build Large Matrix with all camera angles
Subtract the mean from every row
Compute the SVD of the mean less matrix scaled by $\frac{1}{n-1}$
Plot the Variances and the Principle Components
```

4 Computational Results

Ideal Case (case 1):
For this case, there is one component that corresponds to 85% of the energy in Figure 1. The rest of the components have low energy. This makes sense in this case since there is one direction that the pendulum in bouncing. We can see that the first Principle Component is periodic too which relates to the system since the paint can oscillates in Figure 2.

Noisy case (case 2):
We see that there is one major principle components in this case taking up 63% of the energy in Figure 4. Our first principle component shows more noise as it oscillates when compared to the first case. This is due to the added noise in the system due to the camera movement. The second principle component takes up 20% of the energy. This component might be the noise in the system. Again in figure 3, we see oscillatory motion which reflects the cans up and down movement.

Horizontal displacement (case 3):
We see 4 principal components that seem to capture significant amounts of energy in Figure 6. This would highlight the fact that there is horizontal and vertical motion. We can see that it is a little more difficult

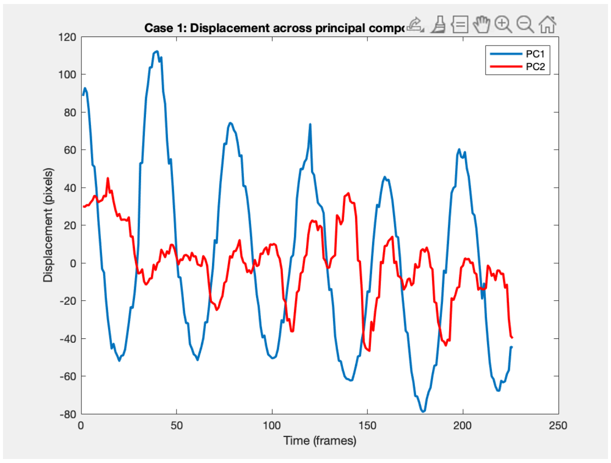

Figure 1: Plots of principle components in case 1

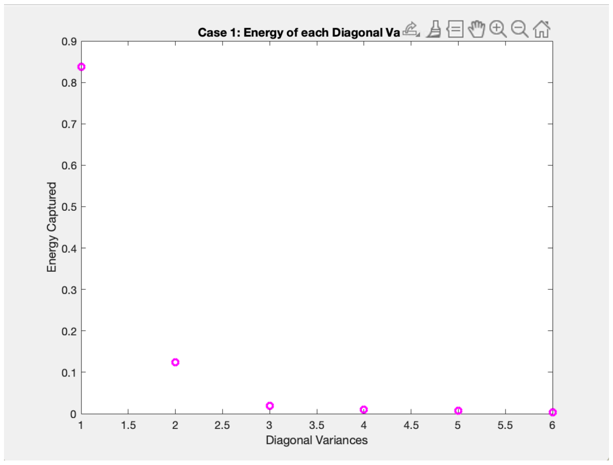

Figure 2: Plot of variances of principal components in case 1

to pull out which component relates to which part of the motion.In figure 5, we can see that there remains oscillatory behavior.

Horizontal displacement and rotation (case 4):
Although our first principle component is massive compared to the rest in Figure 8, we can see the first two principle components are all periodic. This highlights the fact that PCA can identify complex behaviors such as vertical, horizontal, and rotational motion.

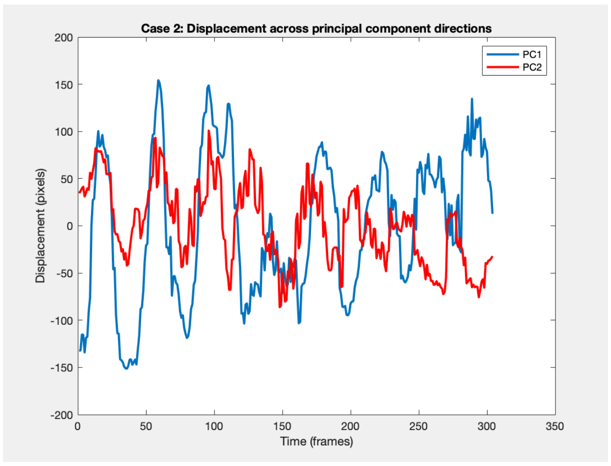

Figure 3: Plots of principle components in case 2

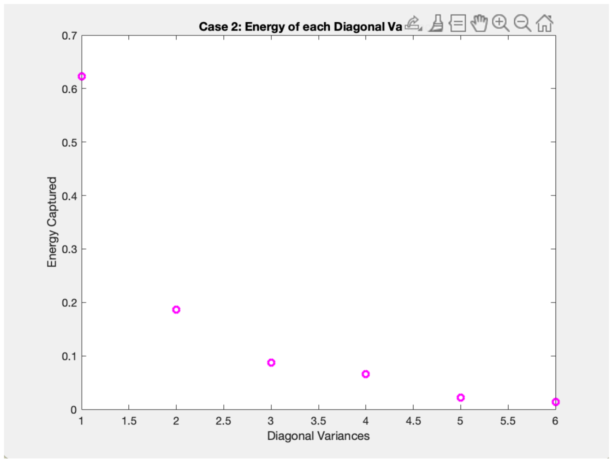

Figure 4: Plot of variances of principal components in case 2

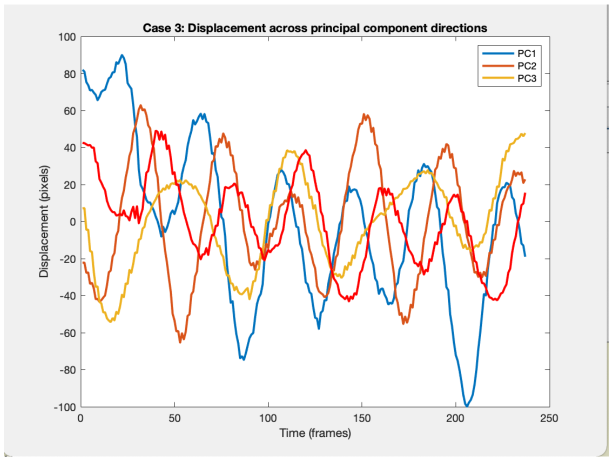

Figure 5: Plots of principle components in case 3

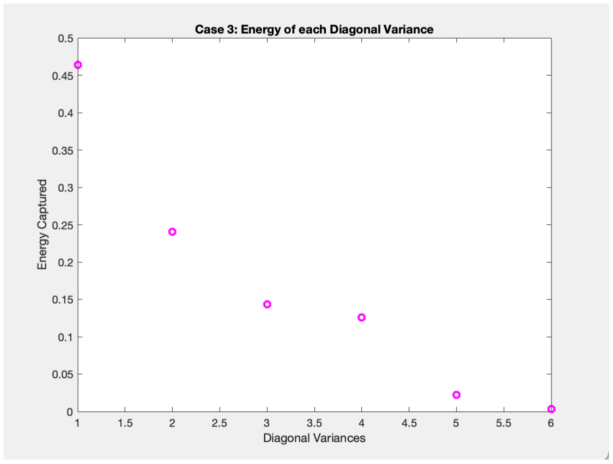

Figure 6: Plot of variances of principal components in case 3

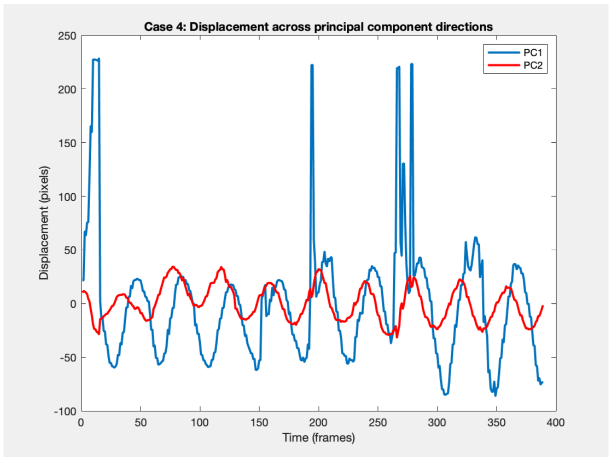

Figure 7: Plots of principle components in case 4

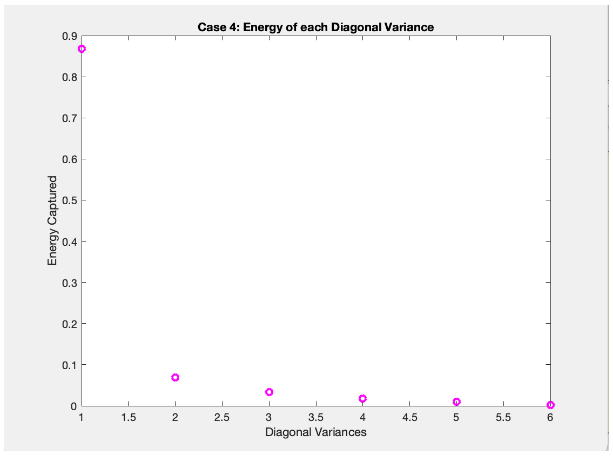

Figure 8: Plot of variances of principal components in case 4

5 Summary and Conclusions

Overall all, we see the power of PCA and its ability to detect core components of a system. PCA is able to see through noise and pick up complex relationships as well. Our investigation has showed us how PCA works on a fundamental level and highlighted some of its weaknesses such as not being very interpretable. It is difficult to pinpoint exactly which component relates to which part of the system.

Appendix A MATLAB Functions

MATLAB functions used and implementation

"diag(X)" : Returns a column vector of the diagonal values of a matrix. "find(X): Returns non zero elements in Matrix X "[i,j,k] = ind2sub(siz,IND)" : Given a index "IND", it returns the "i", "j", and "k" indices, which can then be translated into spacial or frequency domain coordinates.

"max(A)" : Returns maximum values in vector A "mean(A)" : Returns the mean of the vector A. "[M,I] = min(A)" : Returns the minimum value M and the index I of the array.

"rgb2gray(X)" : Converts image to gray scale

"size(X)" : returns dimensions of the matrix X

"svd(X, 'econ') : computes the SVD of the matrix X. "zeros(A,B,C)" : Creates a matrix filled with zeros of size "A x B x C".

Appendix B MATLAB Code

```
clear all; clc; close all;
%%
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);

%%

numFrames1 = size(vidFrames1_1, 4);
means1 = [];

for j = 1:numFrames1
    double1 = double(rgb2gray(vidFrames1_1(:,:,:,j)));
    light = double1 > 240;
    placement1 = find(light);
    [Y1, X1] = ind2sub(size(light),placement1);
    means1 = [means1; mean(Y1), mean(X1)];
end

numFrames2 = size(vidFrames2_1, 4);
means2 = [];

for j = 1:numFrames2
    double2 = double(rgb2gray(vidFrames2_1(:,:,:,j)));
    light = double2 > 240;
    placement2 = find(light);
    [Y2, X2] = ind2sub(size(light),placement2);
    means2 = [means2; mean(Y2), mean(X2)];
end


numFrames3 = size(vidFrames3_1, 4);
means3 = [];

for j = 1:numFrames3
    double3 = double(rgb2gray(vidFrames3_1(:,:,:,j)));
    light = double3 > 240;
    placement3 = find(light);
    [Y3, X3] = ind2sub(size(light),placement3);
    means3 = [means3; mean(Y3), mean(X3)];
end

data2 = means2(1:length(means1), :);
data3 = means3(1:length(means1), :);
data1 = means1;

%%
bigMatrix = [data1';data2';data3'];

[m,n]=size(bigMatrix); % compute data size
mn=mean(bigMatrix,2); % compute mean for each row
bigMatrixmeanless = bigMatrix-repmat(mn,1,n); % subtract mean

[u,s,v]=svd(bigMatrixmeanless'/sqrt(n-1), 'econ'); % perform the SVD
lambda=diag(s).^2; % produce diagonal variances

Y= bigMatrixmeanless' * v; % produce the principal components projection
```

```matlab
clear all; clc; close all;
%%
load('cam1_2.mat'); load('cam2_2.mat'); load('cam3_2.mat')

%%
numFrames1 = size(vidFrames1_2,4);
numFrames2 = size(vidFrames2_2,4);
numFrames3 = size(vidFrames3_2,4);


%%

numFrames1 = size(vidFrames1_2, 4);
means1 = [];
%Play video
width = 50;
filter = zeros(480,640);
filter(300-2.6*width:1:300+2.6*width, 350-width:1:350+2*width) = 1;

for j = 1:numFrames1
    double1 = double(rgb2gray(vidFrames1_2(:,:,:,j)));
    double1 = double1 .* filter;
    light = double1 > 250;
    placement1 = find(light);
    [Y1, X1] = ind2sub(size(light),placement1);
    means1 = [means1; mean(Y1), mean(X1)];
end

numFrames2 = size(vidFrames2_2, 4);
means2 = [];

width = 50;
filter = zeros(480,640);
filter(250-4*width:1:250+4.5*width, 290-2.5*width:1:290+2.7*width) = 1;

for j = 1:numFrames2
    double2 = double(rgb2gray(vidFrames2_2(:,:,:,j)));
    double2 = double2 .* filter;
    light = double2 > 250;
    placement2 = find(light);
    [Y2, X2] = ind2sub(size(light),placement2);
    means2 = [means2; mean(Y2), mean(X2)];
end


width = 50;
filter = zeros(480,640);
filter(250-1*width:1:250+2.6*width, 360-2.5*width:1:360+2.7*width) = 1;
numFrames3 = size(vidFrames3_2, 4);
means3 = [];

for j = 1:numFrames3
    double3 = double(rgb2gray(vidFrames3_2(:,:,:,j)));
    double3 = double3 .* filter;
    light = double3 > 250;
    placement3 = find(light);
    [Y3, X3] = ind2sub(size(light),placement3);
    means3 = [means3; mean(Y3), mean(X3)];
end
%%
means1 = rmmissing(means1, 1);
```

```
clear all; clc; close all;
%%
load('cam1_3.mat'); load('cam2_3.mat'); load('cam3_3.mat')
%%


%%
numFrames1 = size(vidFrames1_3,4);
numFrames2 = size(vidFrames2_3,4);
numFrames3 = size(vidFrames3_3,4);


%%

numFrames1 = size(vidFrames1_3, 4);
means1 = [];

width = 50;
filter = zeros(480,640);
filter(300-2.6*width:1:300+2.6*width, 350-width:1:350+2*width) = 1;

for j = 1:numFrames1
    double1 = double(rgb2gray(vidFrames1_3(:,:,:,j)));
    double1 = double1 .* filter;
    light = double1 > 240;
    placement1 = find(light);
    [Y1, X1] = ind2sub(size(light),placement1);
    means1 = [means1; mean(Y1), mean(X1)];
end

numFrames2 = size(vidFrames2_3, 4);
means2 = [];

width = 50;
filter = zeros(480,640);
filter(250-4*width:1:250+4.5*width, 290-2.5*width:1:290+2.7*width) = 1;

for j = 1:numFrames2
    double2 = double(rgb2gray(vidFrames2_3(:,:,:,j)));
    double2 = double2 .* filter;
    light = double2 > 240;
    placement2 = find(light);
    [Y2, X2] = ind2sub(size(light),placement2);
    means2 = [means2; mean(Y2), mean(X2)];
end

width = 50;
filter = zeros(480,640);
filter(250-1*width:1:250+2.6*width, 360-2.5*width:1:360+2.7*width) = 1;


numFrames3 = size(vidFrames3_3, 4);
means3 = [];

for j = 1:numFrames3
    double3 = double(rgb2gray(vidFrames3_3(:,:,:,j)));
    double3 = double3 .* filter;
    light = double3 > 240;
    placement3 = find(light);
    [Y3, X3] = ind2sub(size(light),placement3);
    means3 = [means3; mean(Y3), mean(X3)];
```

```
clear all; clc;
%%
load('cam1_4.mat'); load('cam2_4.mat'); load('cam3_4.mat')

%%
numFrames1 = size(vidFrames1_4,4);
numFrames2 = size(vidFrames2_4,4);
numFrames3 = size(vidFrames3_4,4);


%%

numFrames1 = size(vidFrames1_4, 4);
means1 = [];

for j = 1:numFrames1
    double1 = double(rgb2gray(vidFrames1_4(:,:,:,j)));
    light = double1 > 240;
    placement1 = find(light);
    [Y1, X1] = ind2sub(size(light),placement1);
    means1 = [means1; mean(Y1), mean(X1)];
end

numFrames2 = size(vidFrames2_4, 4);
means2 = [];

for j = 1:numFrames2
    double2 = double(rgb2gray(vidFrames2_4(:,:,:,j)));
    light = double2 > 240;
    placement2 = find(light);
    [Y2, X2] = ind2sub(size(light),placement2);
    means2 = [means2; mean(Y2), mean(X2)];
end


numFrames3 = size(vidFrames3_4, 4);
means3 = [];

for j = 1:numFrames3
    double3 = double(rgb2gray(vidFrames3_4(:,:,:,j)));
    light = double3 > 240;
    placement3 = find(light);
    [Y3, X3] = ind2sub(size(light),placement3);
    means3 = [means3; mean(Y3), mean(X3)];
end


%%
means1 = rmmissing(means1, 1);
means2 = rmmissing(means2, 1);
means3 = rmmissing(means3, 1);
%%
data2 = means2(1:length(means3), :);
data3 = means3;
data1 = means1(1:length(means3), :);
%%
bigMatrix = [data1';data2';data3'];
```

```

[m,n]=size(bigMatrix); % compute data size
mn=mean(bigMatrix,2); % compute mean for each row
sub = repmat(mn,1,n);
```