AMATH 482 Homework 3

Gg Tran

February 25, 2019

Abstract

We were presented with 4 datasets. Each dataset consisted of a video clip of a moving paint can, filmed by 3 different cameras. Each camera may present noise that affect the recording of the can's movement. Using the Principle Component Analysis (PCA), we tested to see if the PCA would be able to get the principle components that would most explain the variance in the dataset. While we were able to obtain the principle component that represented the can't oscillation in test 1 and test 2, we failed to obtain the principle component representing the can't pendulum motion in test 3 and test 4. However, we believe this failure was due to our data preprocessing, and not due to the failure of the PCA.

I. Introduction and Overview

The Principle Component Analysis is a technique to perform data dimensionality reduction. Many variables in a dataset are redundant, having high correlation with one another. Thus, reducing the dimension of the data can help discover the underlying trends. The Principal Component Analysis finds the maximal variances in the data and project them onto a smaller dimensional space.

II. Theoretical Background

The Principle Component Analysis (PCA) uses eigenvalue and eigenvector diagonalization technique. One way to do eigenvector diagonalization is to form the covariance matrix. Another, and better, method is to use the Singular Value Decomposition (SVD). The SVD guarantees that one can always the perform the SVD on any given matrix $\mathbf{A}$. The SVD takes the form of

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

where $\mathbf{U} \in \mathbb{C}^{m\times m}$ and $\mathbf{V} \in \mathbb{C}^{n\times n}$ are unitary matrices. $\mathbf{V}^*$ is the conjugate transpose of $\mathbf{V}$. $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ is a diagonal matrix. [1]

The SVD helps reduce redundancy in data as well as identifying large singular values (variance). The SVD/PCA assumes linearity and that larger variance means more important dynamics [1]. Lastly, the PCA assumes that one can construct the distribution of the data with two parameter: mean and variance. [1]

III. Algorithm Implementation and Development

Data preprocessing

Our dataset consisted of x-y coordinates of the paint can over time. To obtain the coordinates, for every video, we followed the subsequence steps: Converted all video frames into black and white → Picked out the shape of the paint can by picking matrix entries that were larger than a certain brightness threshold (e.g > 240). By this point, the shape and movement of the can could be seen as white markings on a black background → Constructed a "masking matrix" **maskMat** consisting of 0 and 1 entries. Multiplying each video frame with the masking matrix effectively covered all the white-markings regions that were not the location of the paint can. Each video frame yielded vectors of x and y coordinates of the can, so we took the average the x and y coordinates. To make sure the resulting means lied on the can, we checked by plotting the point coordinates onto the original video. This method accuratedly located the position of the can throughout the video (figure a). The resulting means were saved into vectors $\mathbf{x}$ and $\mathbf{y}$.

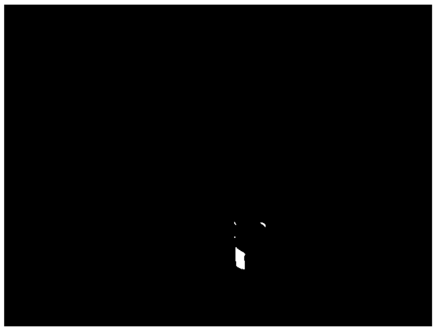

Figure a: Left) A video frame that has been converted to grey scale. After filtering out frame entries with a certain light threshold, as well as covering redundant parts of the frame, we are left with the picture of the can. Right) A video frame in full color. The red dot was plotted to check whether the coordinates we obtained were the correct coordinates of the can.

Performing the PCA

To perform the PCA, we needed to arrange our data into one matrix X where the rows were the number of measurement types, and the columns were the time length. Because each camera recorded over different time length, we truncated the column length of **X** at the minimum recording time between 3 cameras. Thus, for each test, we got the data matrix **X**, with 6 rows corresponding to 6 measurement types. We performed normalization for each row by subtracting each row entry with the mean of that row.

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{y}_1 \\ \mathbf{x}_2 \\ \mathbf{y}_2 \\ \mathbf{x}_3 \\ \mathbf{y}_3 \end{bmatrix}$$

We obtained the principle axes of the data via the Singular Value Decomposition. Since we sampled 1 scence using 3 cameras (i.e $n = 3$), we corrected for redundancy by taking the degree of freedom $n - 1$. We projected **X** onto the principle axes. Y is the matrix representing 6 principle components (PC).

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

Y = U'*X;
```

IV. Computational Results

For test 1, the PCA was able to extract the principle component (the simple harmotic oscillation). The algorithm managed to extract the PC for test 2 even with noises (camera shake and different orientations of the recording) (Figure 1). However, for both test 3 and test 4, the PCA could only extract the simple harmonic motion of the paint can, as one can see by the single largest PC in test 3 and test 4 (Figure 2).

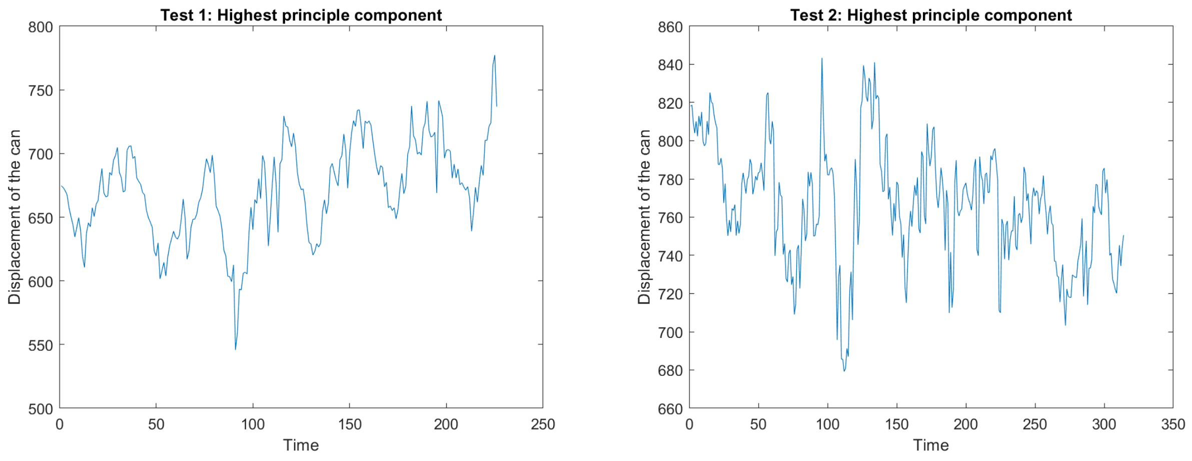

Figure 1: Test 1 and test 2 each yielded one largest PC. This result is expected since the paint can is only moving in simple harmonic motion.

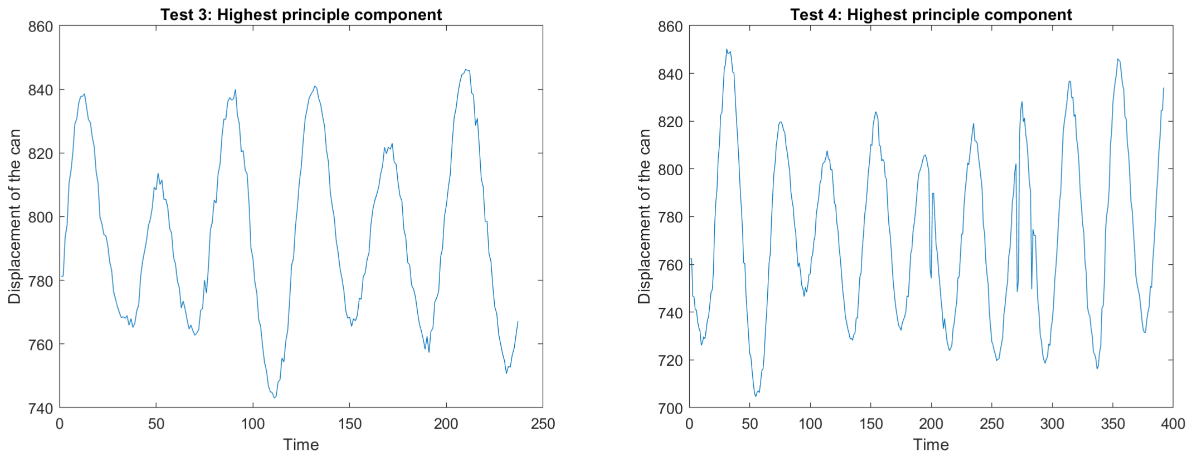

Figure 2: Test 3 and test 4 each also yielded one largest PC. This result is faulty since the paint can is moving in both harmonic motion and pendulum motion.

V. Summary and Conclusions

We were able to extract the motion of the paint can in test 1 and test 2. The PCA performed well even with quite a bit noise in our measurements. However, for test 3 and test 4, there were both simple harmonic motion and pendulum motion. We were only able to extract out the former. This faulty result is due to our method of tracking the paint can. While our tracking worked well for test 1 and test 2, for the last two tests, even though the tracking method correctly located the can, it could not track the pendulum motion. As a result, there were only one prominent principle component in test 3 and test 4.

Appendix A MATLAB functions used

[U, S, V] = svd(): Pulls out the diagonal matrix along with the two unitary matrices U and V.

Appendix B MATLAB codes

Test 1

```
clear all; close all; clc
load('cam1_1.mat')
cam = 1;
a = size(vidFrames1_1);
for k = 1:a(4)
    mov(k).cdata = vidFrames1_1(:,:,:,k);
    mov(k).colormap = [];
end

xCut = a(2)/2;    %col
masking = zeros(a(1), xCut);
nonMask = ones(a(1), xCut);
maskMat = horzcat(masking, nonMask);

frames = a(4);
x = zeros(3, 300)';
y = zeros(3, 300)';
for k = 1:frames
    A = double(rgb2gray(vidFrames1_1(:,:,:,k)));
    A = A >= 255;
    %A = A .* maskMat;
  %  imshow(A)
    [row, col] = find(A == max(A(:)));
    x(k, 1) = mean(col);
    y(k, 1) = mean(row);
end

% %CAM 2
load('cam2_1.mat')
cam = 2;
a = size(vidFrames2_1);
for k = 1:a(4)
    mov(k).cdata = vidFrames2_1(:,:,:,k);
    mov(k).colormap = [];
end

xCut = a(2)/2;    %col
% masking = zeros(a(1), xCut);
% nonMask = ones(a(1), xCut);
% maskMat = horzcat(masking, nonMask);

frames = a(4);

for k = 1:frames
%         X=frame2im(mov(k));
%     subplot(2,1, 1) ,imshow(X); drawnow
    A = double(rgb2gray(vidFrames2_1(:,:,:,k)));
    A = A >= 252;
%         subplot(2,1, 2) ,imshow(A); drawnow
%     A = A .* maskMat;
%     imshow(A);
    [row, col] = find(A == max(A(:)));
    x(k, cam) = mean(col);
    y(k, cam) = mean(row);
end


%CAM 3:
load('cam3_1.mat')
cam = 3;
a = size(vidFrames3_1);
for k = 1:a(4)
    mov(k).cdata = vidFrames3_1(:,:,:,k);
    mov(k).colormap = [];
end

xCut = a(2)/2;    %col
masking = zeros(a(1), xCut-50);
nonMask = ones(a(1), xCut + 50);
maskMat = horzcat(masking, nonMask);

frames = a(4);

for k = 1:a(4)
    mov(k).cdata = vidFrames3_1(:,:,:,k);
    mov(k).colormap = [];
end

for k = 1:frames
%     A=frame2im(mov(k));
%    subplot(2,1, 1) ,imshow(A); drawnow
   A = double(rgb2gray(vidFrames3_1(:,:,:,k)));
   A= A >= 250;
    A = A .* maskMat;
% subplot(2,1, 2)
%     imshow(A)
    [row, col] = find(A == max(A(:)));
    x(k, cam) = mean(col);
    y(k, cam) = mean(row);
end

minSize = 226;
X = [x(:,1)'; y(:, 1)'; x(:,2)'; y(:, 2)'; x(:,3)'; y(:, 3)'];
X = X(:, 1:minSize);
[row, col] = size(X);
XSubtracted = zeros(row,col);
for rows = 1:row
    XSubtracted(rows) = X(rows) - mean(X(rows));
end

n = 3
%U:mm, V: nn, S:mn
[U, S, V] = svd(X/sqrt(n - 1));
Y = U'*X;
```

Test 2

```
clear all; close all; clc
load('cam1_2.mat')
cam = 1;
a = size(vidFrames1_2);
for k = 1:a(4)
    mov(k).cdata = vidFrames1_2(:,:,:,k);
    mov(k).colormap = [];
end

xCut = a(2)/2;    %col
masking = zeros(a(1), xCut-50);
nonMask = ones(a(1), xCut + 50);
maskMat = horzcat(masking, nonMask);

frames = a(4);

for k = 1:a(4)
    mov(k).cdata = vidFrames1_2(:,:,:,k);
    mov(k).colormap = [];
end

for k = 1:frames
%     X=frame2im(mov(k));
%    subplot(2,1, 1) ,
%    imshow(X); drawnow
   A = double(rgb2gray(vidFrames1_2(:,:,:,k)));
   A= A >= 250;
    A = A .* maskMat;
%     subplot(2,1, 2)
%      imshow(A); drawnow
     [row, col] = find(A == max(A(:)));
     x(k, cam) = mean(col);
     y(k, cam) = mean(row);
end


%CAM 2
load('cam2_2.mat')
cam = 2;
a = size(vidFrames2_2);
for k = 1:a(4)
    mov(k).cdata = vidFrames2_2(:,:,:,k);
    mov(k).colormap = [];
end

xCut = a(2)/2;    %col
masking = zeros(a(1)/3, a(2));
nonMask = ones(a(1)*(2/3), a(2));
maskMat = vertcat(masking, nonMask);

frames = a(4);

for k = 1:a(4)
```

```
    mov(k).cdata = vidFrames2_2(:,:,:,k);
    mov(k).colormap = [];
end

for k = 1:frames
%     X=frame2im(mov(k));
%    subplot(2,1, 1) ,
%    imshow(X); drawnow
   A = double(rgb2gray(vidFrames2_2(:,:,:,k)));
   A= A >= 250;
     A = A .* maskMat;
%      subplot(2,1, 2)
%      imshow(A); drawnow
     [row, col] = find(A == max(A(:)));
     x(k, cam) = mean(col);
     y(k, cam) = mean(row);
end

%CAM 3
load('cam3_2.mat')
cam = 3;
a = size(vidFrames3_2);
for k = 1:a(4)
    mov(k).cdata = vidFrames3_2(:,:,:,k);
    mov(k).colormap = [];
end

xCut = a(2)/2;    %col
masking = zeros(a(1), xCut);
nonMask = ones(a(1), xCut);
maskMat = horzcat(masking, nonMask);

frames = a(4);

for k = 1:a(4)
    mov(k).cdata = vidFrames3_2(:,:,:,k);
    mov(k).colormap = [];
end

for k = 1:frames
    X=frame2im(mov(k));
%    subplot(2,1, 1) ,
%    imshow(X); drawnow
   A = double(rgb2gray(vidFrames3_2(:,:,:,k)));
   A= A >= 250;
      A = A .* maskMat;
%      subplot(2,1, 2)
%      imshow(A); drawnow
     [row, col] = find(A == max(A(:)));
     x(k, cam) = mean(col);
     y(k, cam) = mean(row);
end
```

```
minSize = 314;
X = [x(:,1)'; y(:, 1)'; x(:,2)'; y(:, 2)'; x(:,3)'; y(:, 3)'];
X = X(:, 1:minSize);
[row, col] = size(X);
XSubtracted = zeros(row,col);
for rows = 1:row
    XSubtracted(rows) = X(rows) - mean(X(rows));
end

%U:mm, V: nn, S:mn
[U, S, V] = svd(X/sqrt(2));
Y = U'*X;
```

Test 3

```
clear all; close all; clc
load('cam1_2.mat')
cam = 1;
a = size(vidFrames1_2);
for k = 1:a(4)
    mov(k).cdata = vidFrames1_2(:,:,:,k);
    mov(k).colormap = [];
end

xCut = a(2)/2;    %col
masking = zeros(a(1), xCut-50);
nonMask = ones(a(1), xCut + 50);
maskMat = horzcat(masking, nonMask);

frames = a(4);

for k = 1:a(4)
    mov(k).cdata = vidFrames1_2(:,:,:,k);
    mov(k).colormap = [];
end

for k = 1:frames
%     X=frame2im(mov(k));
%    subplot(2,1, 1) ,
%    imshow(X); drawnow
   A = double(rgb2gray(vidFrames1_2(:,:,:,k)));
   A= A >= 250;
    A = A .* maskMat;
%     subplot(2,1, 2)
%      imshow(A); drawnow
     [row, col] = find(A == max(A(:)));
     x(k, cam) = mean(col);
     y(k, cam) = mean(row);
end


%CAM 2
load('cam2_2.mat')
cam = 2;
```

```
a = size(vidFrames2_2);
for k = 1:a(4)
    mov(k).cdata = vidFrames2_2(:,:,:,k);
    mov(k).colormap = [];
end

xCut = a(2)/2;    %col
masking = zeros(a(1)/3, a(2));
nonMask = ones(a(1)*(2/3), a(2));
maskMat = vertcat(masking, nonMask);

frames = a(4);

for k = 1:a(4)
    mov(k).cdata = vidFrames2_2(:,:,:,k);
    mov(k).colormap = [];
end

for k = 1:frames
%     X=frame2im(mov(k));
%    subplot(2,1, 1) ,
%    imshow(X); drawnow
   A = double(rgb2gray(vidFrames2_2(:,:,:,k)));
   A= A >= 250;
     A = A .* maskMat;
%      subplot(2,1, 2)
%      imshow(A); drawnow
     [row, col] = find(A == max(A(:)));
     x(k, cam) = mean(col);
     y(k, cam) = mean(row);
end

%CAM 3
load('cam3_2.mat')
cam = 3;
a = size(vidFrames3_2);
for k = 1:a(4)
    mov(k).cdata = vidFrames3_2(:,:,:,k);
    mov(k).colormap = [];
end

xCut = a(2)/2;    %col
masking = zeros(a(1), xCut);
nonMask = ones(a(1), xCut);
maskMat = horzcat(masking, nonMask);

frames = a(4);

for k = 1:a(4)
    mov(k).cdata = vidFrames3_2(:,:,:,k);
    mov(k).colormap = [];
end
```

```
for k = 1:frames
    X=frame2im(mov(k));
%    subplot(2,1, 1) ,
%    imshow(X); drawnow
   A = double(rgb2gray(vidFrames3_2(:,:,:,k)));
   A= A >= 250;
      A = A .* maskMat;
%      subplot(2,1, 2)
%      imshow(A); drawnow
     [row, col] = find(A == max(A(:)));
     x(k, cam) = mean(col);
     y(k, cam) = mean(row);
end

minSize = 314;
X = [x(:,1)'; y(:, 1)'; x(:,2)'; y(:, 2)'; x(:,3)'; y(:, 3)'];
X = X(:, 1:minSize);
[row, col] = size(X);
XSubtracted = zeros(row,col);
for rows = 1:row
    XSubtracted(rows) = X(rows) - mean(X(rows));
end

%U:mm, V: nn, S:mn
[U, S, V] = svd(X/sqrt(2));
Y = U'*X;
```

Test 4

```
clear all; close all; clc
load('cam1_4.mat')
cam = 1;
a = size(vidFrames1_4);
for k = 1:a(4)
    mov(k).cdata = vidFrames1_4(:,:,:,k);
    mov(k).colormap = [];
end

xCut = a(2)/2;    %col
masking = zeros(a(1)/2, a(2));
nonMask = ones(a(1)/2, a(2));
maskMat = vertcat(masking, nonMask);

frames = a(4);

for k = 1:a(4)
    mov(k).cdata = vidFrames1_4(:,:,:,k);
    mov(k).colormap = [];
end

for k = 1:frames
     X=frame2im(mov(k));
%     subplot(2,1, 1) ,
%     imshow(X); drawnow
%     hold on
   A = double(rgb2gray(vidFrames1_4(:,:,:,k)));
   A= A >= 240;
      A = A .* maskMat;
%     subplot(2,1, 2)
     imshow(A); drawnow
     [row, col] = find(A == max(A(:)));
     x(k, cam) = mean(col)+20;
     y(k, cam) = mean(row)-20;
%      plot(x(k,cam), y(k, cam), 'o', 'color','r','LineWidth', 2); drawnow
k
end

CAM 2
load('cam2_4.mat')
cam = 2;
a = size(vidFrames2_4);
for k = 1:a(4)
    mov(k).cdata = vidFrames2_4(:,:,:,k);
    mov(k).colormap = [];
end

xCut = a(2)/2;    %col
masking = zeros(a(1)/3, a(2));
nonMask = ones(a(1)*(2/3), a(2));
maskMat = vertcat(masking, nonMask);

frames = a(4);

for k = 1:a(4)
    mov(k).cdata = vidFrames2_4(:,:,:,k);
    mov(k).colormap = [];
end

for k = 1:frames
%     X=frame2im(mov(k));
%    subplot(2,1, 1) ,
%    imshow(X); drawnow
   A = double(rgb2gray(vidFrames2_4(:,:,:,k)));
   A= A >= 250;
       A = A .* maskMat;
%     subplot(2,1, 2)
%      imshow(A); drawnow
     [row, col] = find(A == max(A(:)));
     x(k, cam) = mean(col);
     y(k, cam) = mean(row);
end
%
%
% %CAM 3
load('cam3_4.mat')
cam = 3;
a = size(vidFrames3_4);
```

```
for k = 1:a(4)
    mov(k).cdata = vidFrames3_4(:,:,:,k);
    mov(k).colormap = [];
end

xCut = a(2)/2;    %col
masking = zeros(a(1), xCut);
nonMask = ones(a(1), xCut);
maskMat = horzcat(masking, nonMask);

frames = a(4);

for k = 1:a(4)
    mov(k).cdata = vidFrames3_4(:,:,:,k);
    mov(k).colormap = [];
end

for k = 1:frames
%     X=frame2im(mov(k));
%    subplot(2,1, 1) ,
%    imshow(X); drawnow
   A = double(rgb2gray(vidFrames3_4(:,:,:,k)));
   A= A >= 240;
     A = A .* maskMat;
%     subplot(2,1, 2)
%      imshow(A); drawnow
     [row, col] = find(A == max(A(:)));
     x(k, cam) = mean(col);
     y(k, cam) = mean(row);
end

minSize = 392;
X = [x(:,1)'; y(:, 1)'; x(:,2)'; y(:, 2)'; x(:,3)'; y(:, 3)'];
X = X(:, 1:minSize);
[row, col] = size(X);
XSubtracted = zeros(row,col);
for rows = 1:row
    XSubtracted(rows) = X(rows) - mean(X(rows));
end

%U:mm, V: nn, S:mn
[U, S, V] = svd(X/sqrt(2));
Y = U'*X;
```