AMATH 582 Homework 3

Tomas Perez

February 24, 2021

Abstract

In this paper, various test cases are explored using Principal Component Analysis. Three cameras are placed in different spots to film an object in movement, in this case it is a paint can on a bungee cable. Four test cases are examined: 1) simple harmonic motion in one direction, 2) simple harmonic motion in one direction with the introduction of noise, 3) simple harmonic motion and pendulum motion, 4) simple harmonic motion and pendulum motion with rotation. X and Y coordinates for each camera are extracted, then Simple Value Decomposition is used to get the principal components.

1 Introduction and Overview

The experiment involves analyzing the movement of a paint can on a bungee cable. Three cameras are placed around the paint can filming at different angles. The data from the camera is extracted into X and Y movement of time (camera frames). Four test cases are used to illustrate various aspects of Principal Component Analysis (PCA).

- Ideal case: Simple harmonic mostion in the Z direction. The paint can is moved up and down with minimal movement in the X and Y directions.
- Noisy case: Repeat of the Ideal case, but with camera shake added to the video recording. The camera shake introduces noise to the system.
- Horizontal Displacement: Simple harmonic motion in the Z direction and pendulum motion in the XY-plane. The paint can is released off center, producing movement in all directions.
- Horizontal Displacement and Rotation: Repeat of the Horizontal Displacement case, but now rotation is added to the paint can.

2 Theoretical Background

Singular Value Decomposition (SVD) is linear algebra method for factorization of a transformation matrix A by identifying orthonormal bases U and V and a diagonal component of singular values. Applying the transformation matrix A to a vector x changes the magnitude and direction of the vector. SVD decomposes the transformation matrix A into a two rotational matrices U and V, and a scaling matrix Σ.

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

Where U is a $m \times n$ orthonormal matrix, Σ is a $n \times n$ diagonal matrix of singular values, and V is a $n \times n$ unitary matrix. This is called the reduced-form SVD of A. By defining the transformed variable $Y = U^*X$, we can calculate the covariance of Y.

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

In comparison with normal eigendecomposition, we could see that Σ^2 is equivalent to λ. The covariance of Y is the diagonal matrix. By taking a standardized X and scaling it by $\frac{1}{n-1}$ we can get the reduced-form

SVD of X^T. The columns of U are movement of the paint can along the PCA modes, and the columns of V are the horizontal and vertical directions of the PCA modes.

$$X^T = U\Sigma V^* \tag{2}$$

3 Algorithm Implementation and Development

For each test, the process can be broken into two parts: 1) tracking the paint can's movements, and 2) principal component analysis.

```
Algorithm 1: Movement Tracking
    Import data from video recording
    Get the height, width, and frames of the video recording
    Set window size ($\tau$) and step interval ($tslide$)
    for $j = 1 : frames$ do
        convert the image at each frame to grayscale
        Isolate the paint can in the video by making areas around the can equal to zero
        Take the max values and their indices
        Use the indices to construct vectors of X and Y movement
    end for
    Repeat for each camera
    Add X and Y vectors gathered from each camera to a single $6 \times n$ matrix (call it $XY$)
```

```
Algorithm 2: PCA
    Subtract the mean values from the $XY$ matrix
    Multipy $XY$ by it's transposed self and divide by $n - 1$ to get the covariance matrix ($CovMat$)
    Sort the eigenvalues and eigenvector matrix
    Multiply the eigenvector matrix with $CovMat$ to get the projection of the coordinate matrix onto the
    principle component basis
```

4 Computational Results

4.1 Test 1: Ideal Case

Figure 1 shows the horizontal and vertical motion captured by each camera. There is minimal movement in the X and Y directions, with all of the movement being a simple harmonic motion in the Z direction on a 3-D plane. Moving to the 2-D plane, you can see the movement captured in each camera. In cameras 1 and 2, X is the horizontal movement and Y is the verical movement. Camera 3 is rotated, so the actual vertical movement of the paint can is captured along the x-axis. It is clear that all the movement is vertical, with minimal horizontal motion. Figure 2 shows the singular values, the percentage variance explained by the PCA modes, and the the movement in the principal component. The first two PCA modes are significantly more important than the others, explaining roughly 90% of the variance.

Test 1

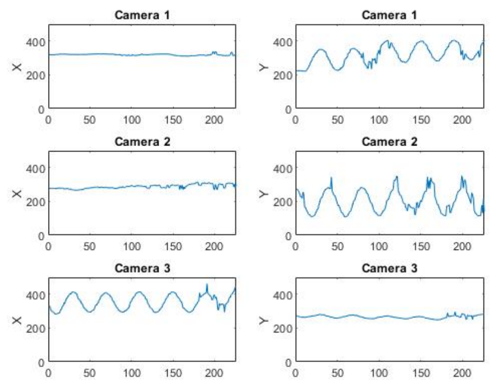

Figure 1: X and Y Positions

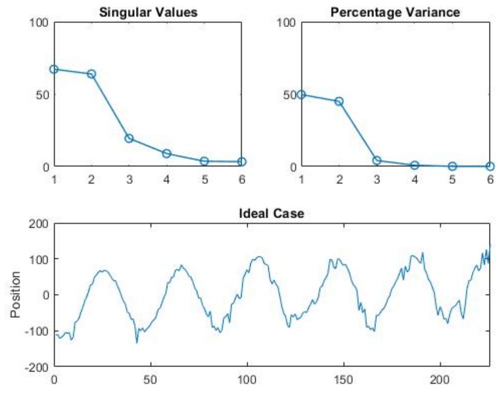

Figure 2: Sigma and Principal Component

Test 2

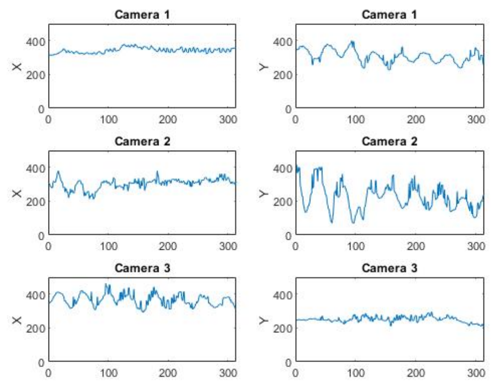

Figure 3: X and Y Positions

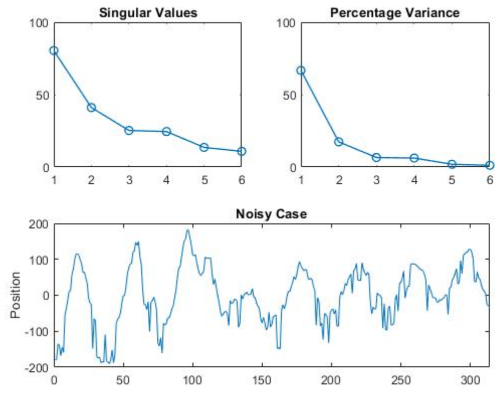

Figure 4: Sigma and Principal Component

4.2 Test 2: Noisy Case

Looking at Figure 3, it's easy to see the increased choppiness of the footage caused by the camera shake. The horizontal movement is significantly less stabe, and more erratic. As expected, we've lost relative smoothness with the introduction of noise to the data. The results are still consistent with the first test, and most of the movement is still found in the vertical direction. However, the impact of noise on the system is clearly evident. The first PCA mode explains more of the variance in this test, compared to the ideal case.

4.3 Test 3: Horizontal Displacement

Figure 5 shows clear indication of horizontal and vertical oscillations. With little noise in the system, the motion isn't as erratic. The smoother oscillations in horizontal movement captured by the cameras can be contributed to the true horizontal movement of the paint can. The percentage variance explained is more distributed among the PCA modes.

Test 3

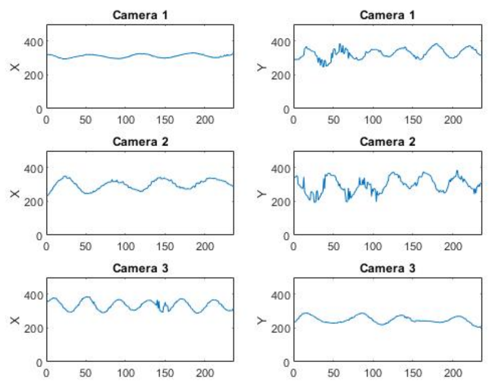

Figure 5: X and Y Positions

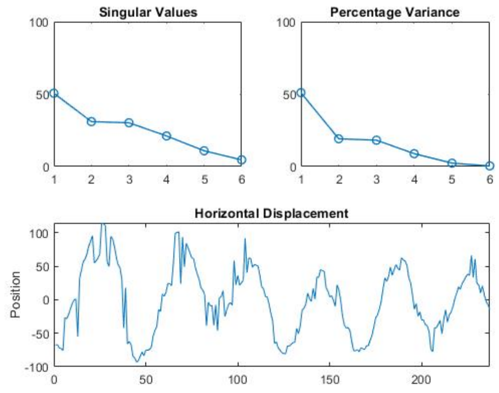

Figure 6: Sigma and Principal Component

Test 4

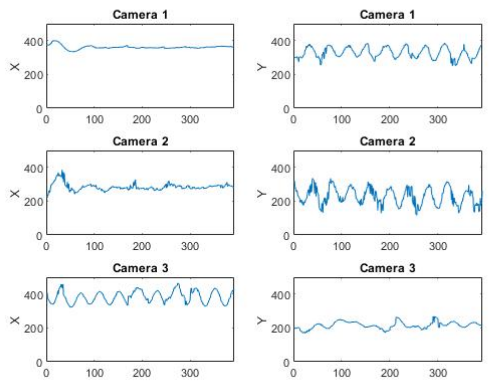

Figure 7: X and Y Positions

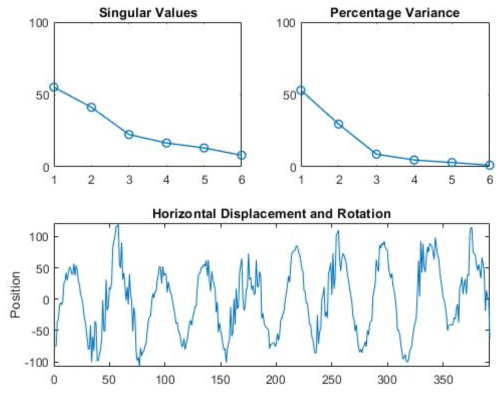

Figure 8: Sigma and Principal Component

4.4 Test 4: Horizontal Displacement with Rotation

Visible in Figure 7, Test 4 had less horizontal motion overall than in Test 3 with most of the horizontal movement occurring early in the video recording. The horizontal motion is not as smooth as in Test 3. The rotation doesn't significantly impact the motion of the paint can that is being tracked in this experiment. It acts more closely to additional noise added to part of the footage. The first two PCA modes explain roughly 85% of the variance.

5 Summary and Conclusions

This study illustrates the usefulness of applying PCA to extract the dynamics of the system created by data collected from different sources. The algorithm implemented successfully tracked the horizontal and vertical motion of the paint can. Principal component analysis was able to identify the harmonic and pendulum motions relatively well. However, the introduction of noise to the system had a clear impact on the results between Test 1 and Test 2. The dynamics of each system were mostly captured by the first two, or three, PCA modes. When the data is relatively clean, PCA does a good job, but of the noisy case could be improved with a better method implemented to clean the data.

Appendix A MATLAB Functions

- `I = rgb2gray(RGB)` onverts the truecolor image RGB to the grayscale image I
- `M = mean(A)` returns the mean of the elements of A along the first array dimension whose size does not equal 1
- `B = repmat(A,n)` returns an array containing n copies of A in the row and column dimensions. The size of B is size(A)*n when A is a matrix
- `[row,col] = ind2sub(V,I)` returns the arrays row and col containing the equivalent row and column subscripts corresponding to the linear indices `I` for a matrix of size `V`
- `[M,I] = max(A)` returns the maximum element of an array and the index into the operating dimension that corresponds to the maximum value of `A`
- `[V,D]=eig(A)` returns diagonal matrix D of eigenvalues and matrix V whose columns are the corresponding right eigenvectors, so that A*V = V*D
- `D=diag(v)` returns a square diagonal matrix with the elements of vector v on the main diagonal
- `B=sort(A)` sorts the elements of A in ascending order
- `[U,S,V]=svd(A)` performs singular value decomposition of matrix A, such that A=U*S*V

Appendix B MATLAB Code

For brevity, the code below is for Test 1) Ideal Case. The code is essentially identical for all four tests, only differing in the size of the graph axes and the areas of each image being blacked out. Other than that, the code is the same for each test case.

```
close all; clear all; clc
load cam1_1.mat; load cam2_1.mat; load cam3_1.mat;

[h,w,rgb,frames]=size(vidFrames1_1);
X1=[]; Y1=[];
for j=1:frames
    img=rgb2gray(vidFrames1_1(:,:,:,j));
    img(:,1:310)=0;
    img(:,390:end)=0;
    img(1:200,:)=0;
    [M,I]=max(img(:));
    [Iy,Ix]=ind2sub(size(img),I);
    X1=[X1 Ix]; Y1=[Y1 Iy];
end

[h,w,rgb,frames]=size(vidFrames2_1);
X2=[]; Y2=[];
for j=1:frames
    img=rgb2gray(vidFrames2_1(:,:,:,j));
    img(:,1:245)=0;
    img(:,350:end)=0;
    img(1:100,:)=0;
    img(370:end,:)=0;
    [M,I]=max(img(:));
    [Iy,Ix]=ind2sub(size(img),I);
    X2=[X2 Ix]; Y2=[Y2 Iy];
end

[h,w,rgb,frames]=size(vidFrames3_1);
X3=[]; Y3=[];
for j=1:frames
    img=rgb2gray(vidFrames3_1(:,:,:,j));
    img(:,1:250)=0;
    img(:,500:end)=0;
    img(1:230,:)=0;
    img(340:end,:)=0;
    [M,I]=max(img(:));
    [Iy,Ix]=ind2sub(size(img),I);
    X3=[X3 Ix]; Y3=[Y3 Iy];
end
```

Listing 1: Tracking

```
Xmin=min([length(X1) length(X2) length(X3)]);
if length(X1) > Xmin
    X1=X1(1:Xmin);
    Y1=Y1(1:Xmin);
end
if length(X2) > Xmin
    X2=X2(1:Xmin);
    Y2=Y2(1:Xmin);
end
if length(X3) > Xmin
    X3=X3(1:Xmin);
    Y3=Y3(1:Xmin);
end
XY=[X1;Y1;X2;Y2;X3;Y3];
figure(1)
for j=1:6
    subplot(3,2,(j)), plot(XY(j,:))
    xlim([0 226]), ylim([0 500])
    if j==1 || j==2
        title('Camera 1')
    elseif j==3 || j==4
        title('Camera 2')
    elseif j==5 || j==6
        title('Camera 3')
    end
    if mod(j,2)==1 %odd
        ylabel('X')
    elseif mod(j,2)==0 %even
        ylabel('Y')
    end
end
[row,col]=size(XY);
mn=mean(XY,2);
XY=XY-repmat(mn,1,col);
CovMat=(1/(col-1))*XY*XY';
[eVec,eVal]=eig(CovMat);
lambda=diag(eVal);
[B,ind]=sort(-1*lambda);
lambda=lambda(ind);
eVec=eVec(:,ind);
PComp=eVec'*XY;
[U,S,V]=svd(XY'/sqrt(col-1),0);
lambda2=diag(S).^2;
sigma =diag(S); PComp2=U*XY;
figure(2)
subplot(2,2,1), plot(sigma,'-o','Linewidth',[1])
axis([1 6 0 100])
title('Singular Values')
subplot(2,2,2), plot(lambda2/sum(lambda)*100,'-o','Linewidth',[1])
axis([1 6 0 100])
title('Percentage Variance')
subplot(2,1,2), plot(PComp(1,:))
xlim([0 226])
ylabel('Position'), title('Ideal Case')
figure(4)
plot(PComp2(1,:))
figure(5)
plot(sigma,'-o','Linewidth',[1])
axis([1 6 0 100])
title('Singular Values')
```