Principal Component Analysis of Video Data

Makenna Barton - bartonmj@uw.edu

February 2020

Abstract This report outlines the techniques and procedures necessary to acquire position data over time from a video, perform Principal Component Analysis and find the energy of each dimension to provide information on how the data varied with each other. After analysis it was clear that the main motion of the paint cans were only in one dimension.

1 Introduction and Overview

Four cases of a paint can's oscillating motion from a spring were recorded at three different camera angles. The four cases were: ideal, noisy, horizontal displacement, and horizontal displacement with rotation. Comparing the motion from each camera for each case gives information about the overall path of the paint can. These readings are technically redundant as they are all tracking the same motion, but performing the Principal Component Analysis will mathematically tell us the redundancy of the data.

2 Theoretical Background

The analysis used in this problem lies within the concepts of the Singular Value Decomposition (SVD). The SVD is a form of factoring a matrix into other matrices and its foundation comes from Eigenvalue Decomposition. The core principles of these techniques are explained in the properties of matrix multiplication and how the data are rotated and stretched through linear transformations.

$$A\vec{v}_1 = \sigma_1\vec{u}_1 A\vec{v}_2 = \sigma_2\vec{u}_2 A\vec{v}_n = \sigma_n\vec{u}_n \tag{1}$$

The reduced SVD is:

$$A = \hat{U}\hat{\Sigma}V^T \tag{2}$$

where $\hat{U}$ and V^T are unitary matrices that contain information on rotation and $\hat{\Sigma}$ is a diagonal matrix describing the stretch in each direction. The columns of $\hat{U}$ are the left singular vectors of A and the columns of V are called the right singular vectors of A. Each non-zero value in the $\hat{\Sigma}$ matrix is a singular value of A and the total number of non-zero singular values is the rank of A. Additional properties of singular values are as followed:

- The singular values are always non-negative
- The singular values are always ordered from largest to smallest (i.e. $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots \geq \sigma_n$)
- Let $r = rank(A)$, then $\{\vec{u}_1, \vec{u}_2, \ldots, \vec{u}_n\}$(the column vectors of U) is a basis for the range of A and the set $\{\vec{v}_{r+1}, \ldots, \vec{v}_n\}$(the column vectors of V) is a basis for the null space of A

The Eigenvalue Decomposition and SVD are related in many ways. The SVD allows the factorization of any $m \times n$ matrix, not only square like Eigenvalue Decomposition. The singular values of A are the square roots of the eigenvalues of $A^T A$. The right singular vectors of A are the eigenvectors of $A^T A$. The left singular vectors of A are the eigenvectors of AA^T.

The Principal Component Analysis (PCA) allows us to apply a meaningful usage of the factorization we get from the SVD. The SVD tells us the dimension and the direction of the principal components of the matrix A. We get information on how much of the data is contained in each dimension, through the singular values, and this is extremely helpful when looking to reduce the number of dimensions of a data set.

We need to understand some aspects of statistics to understand the PCA of a matrix because the principal components have to do with the variance of the data.

$$variance = \sigma^2 = \frac{1}{n}\sum_{k=1}^{n}(a_k - \mu)^2 \tag{3}$$

where μ is the average all the data $\{a_1, a_2, \ldots, n\}$.

The energy of a dimension of data tell how much of the total data is contained in that dimension. The formula of a rank r matrix calculated to the n dimension is:

$$energy = \frac{\sigma_1^2 + \cdots + \sigma_n^2}{\sigma_1^2 + \cdots + \sigma_r^2} \tag{4}$$

Using PCA and understanding the energy of a matrix, you can tell how much of the total data can be captured within the chosen dimension, providing useful information that can lead to representing the data in a lower dimension. Lower dimensional representation of data is very useful when using large data sets.

3 Algorithm Implementation and Development

To begin to analyze the video files provided for this problem, I considered a number of methods to track the position of the paint can over time. My initial thought was to use the flashlight that is attached to the can as a point to track; converting the video to gray scale and tracking the brightest pixel. Some issues that arose with this method are that at times the paint can is rotating or the camera is at an angle where you are unable to see the flashlight, therefore the brightest pixel could not be tracked. There was another situation where the pixels from the light were actually not the brightest on the frame, causing the detected pixel to skew the path of the paint can. After considering many approaches and playing with built in Matlab functions and toolboxes, I decided to take a manual approach to collecting the position data from the videos. I felt this approach would allow the most precision and least error. Though it must be noted, this technique was not free of error. The precision of my eye and accuracy of selecting the correct pixel may have been compromised by trying to collect the data quickly. There were frames where there was a glitch or I twitched and the point recorded was not exact. It is also important to consider the frames where the picture was extremely blurred, either due to camera shake, or low frame resolution. The point collected on those frames have a lower level of accuracy.

When manually following the paint can on its path, I chose to follow its center of mass for the first 3 video cases. In the fourth case, I followed and the attached flashlight to capture the paint can's rotation. Additionally, in the second case, the noisy case, I recorded the data of 2 points per frame: the center of mass of the can, and a point that, in an ideal case, was fixed in the frame (such as a point on the wall, desk, or whiteboard). The reason for this was to be able to track the can's motion, and the motion of the noise through how the stationary point acted. After collecting the data on both points for each camera angle, I subtracted the noisy point from each position for the can, resulting in a smooth path.

To make sure all three camera angle's of the same case of motion were timed together, I took an extra step before recording the points. I watched each video and visually determined the frame at which each video showed the paint can to be at its peak position in oscillation. Starting data collection at that point ensured the timing of each position was related between camera angles.

Once I had the position data at each camera angle for all four cases, I combined each case's data into one matrix. This included making sure the data lasted the same amount of time (same number of frames recorded) and listing both the x and y coordinate vectors for each camera angle, resulting in a matrix with 6 rows for each case.

After collection and organization I was able to perform analysis on the data. First thing I did was take the SVD of each matrix, leaving me with $[U, S, V]$ matrices for each case.

I then used the singular values in the S matrix to calculate the energy of each of each dimension. The energies allowed me to know which dimension contained the majority of the motion of the paint can.

4 Computational Results

The energies of each dimension are the following:

- Case 1: .9883 (rank 1), .9915 (rank 2), .9923 (rank 3), .9926 (rank 4), .9927 (rank 5), .9928 (rank 6)

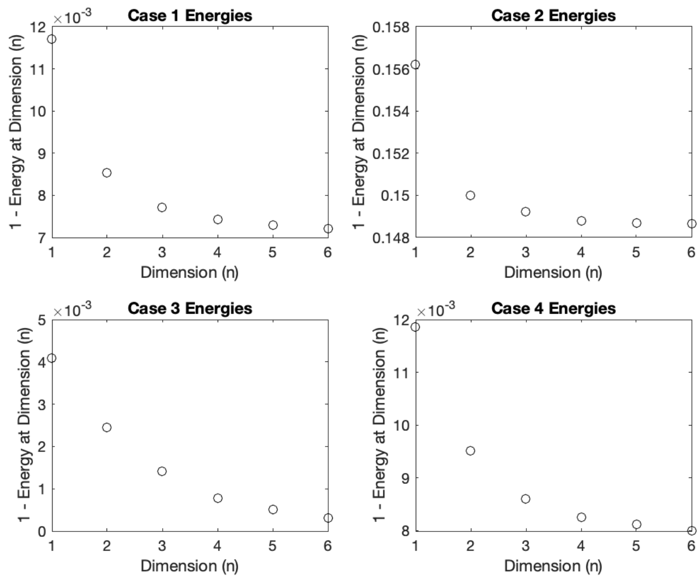

Figure 1: The amount of data captured in each dimension for each case

- Case 2: .8438 (rank 1), .8500 (rank 2), .8508 (rank 3), .8512 (rank 4), .8513 (rank 5), .8514 (rank 6)
- Case 3: .9959 (rank 1), .9975 (rank 2), .9986 (rank 3), .9992 (rank 4), .9995 (rank 5), .9997 (rank 6)
- Case 4: .9881 (rank 1), .9905 (rank 2), .9914 (rank 3), .9917 (rank 4), .9919 (rank 5), .9920 (rank 6)

This tells us how much of the data is contained in each dimension. Further, 98.83% of the data is captured in one dimension for case 1. Case 2, 84.38% of the data is captured in one dimension as well, with a total of 85.14% of the data being captured in 6 dimensions overall. This shows us that almost all the data we had could be measured in one dimension, and the total data captured is not as close to 100% because of the noise in case 2. Case 3 captured 99.86% of the data in 3 dimension and Case 4 captured 99.14% of the data in 3 dimensions.

5 Summary and Conclusions

The PCA us a powerful tool that provides insights that lead to dimension reduction of data. This was useful in our situation of recording the motion of the paint can because it told us about how many directions the can was moving in.

Through collection of the position data from the paint can videos then performing PCA, we were able to see that most of the data for Case 1, the ideal case, was collected in one dimension. nost of the data was collected in one dimension for the Case 2, the noisy case, but overall less data was represented due to the noise. Case 3 and Case 4 were mostly moving in one dimension, but important data was also recorded in the second and third dimensions.

Due to the overwhelming amount of data that was collected in one dimension for all cases tells us that the main motion of the paint can in all situations was in one dimension.

6 Appendix A - MATLAB functions used and brief implementation explanation

`implay()`- Plays back the video data
`ginput()`- Records the coordinates of the points selected through a click on the video frame
`imshow()`- Shows an image
`svd()`- Singular Value Decomposition, outputs the factorization into a left singular vector matrix, a matrix with the singular values, and a matrix with the right singular values

7 Appendix B - MATLAB Codes

Matlab Code

```
%% Case 1
load('cam1_1.mat')
load('cam1_2.mat')
load('cam1_3.mat')
implay(vidFrames1_1)
% Loop to show each frame of the video
numFrames = size(vidFrames1_1,4);
% Found visually
peak_frame = 50;
% numFrames columns starting at peak, 6 rows for the x and y position at each
% camera angle
matrix1 = zeros(6,numFrames - peak_frame);
for j = 1:numFrames - peak_frame
    X = vidFrames1_1(:,:,:,j + peak_frame - 1);
    imshow(X);
    drawnow
    [cam1_1x,cam1_1y] = ginput(1)
    matrix1(1,j) = cam1_1x;
    matrix1(2,j) = cam1_1y;
end
save('matrix1.mat','matrix1')
%%
numFrames = size(vidFrames2_1,4);
peak_frame = 100;
matrix1_2 = zeros(2,numFrames - peak_frame);
for j = 1:numFrames - peak_frame
    X = vidFrames2_1(:,:,:,j + peak_frame - 1);
    imshow(X);
```

```
    drawnow
    [cam2_1x,cam2_1y] = ginput(1)
    matrix1_2(1,j) = cam2_1x;
    matrix1_2(2,j) = cam2_1y;
end
save('matrix1_2.mat','matrix1_2')
%%
numFrames = size(vidFrames3_1,4);
peak_frame = 50;
matrix1_3 = zeros(2,numFrames - peak_frame);
for j = 1:numFrames - peak_frame
    X = vidFrames3_1(:,:,:,j + peak_frame - 1);
    imshow(X);
    drawnow
    [cam3_1x,cam3_1y] = ginput(1)
    matrix1_3(1,j) = cam3_1x;
    matrix1_3(2,j) = cam3_1y;
end
save('matrix1_3.mat','matrix1_3')


%% Case 2 - every other frame to reduce impact from noise
load('cam1_2.mat')
load('cam2_2.mat')
load('cam3_2.mat')
implay(vidFrames1_2)
% Loop to show each frame of the video
numFrames = size(vidFrames1_2,4);
% Found visually
peak_frame = 74;
% numFrames columns starting at peak
matrix2_1 = zeros(4,numFrames - peak_frame);
for j = 1:numFrames - peak_frame
    X = vidFrames1_2(:,:,:,j*2 + peak_frame - 1);
    imshow(X);
    drawnow
    [cam1_2x,cam1_2y] = ginput(2)
    matrix2_1(1:2,j) = cam1_2x;
    matrix2_1(3:4,j) = cam1_2y;
end
save('matrix2_1.mat','matrix2_1')
%%
numFrames = size(vidFrames2_2,4);
peak_frame = 100;
matrix2_2 = zeros(4,numFrames - peak_frame);
for j = 1:numFrames - peak_frame
```

```
    X = vidFrames2_2(:,:,:,j*2 + peak_frame - 1);
    imshow(X);
    drawnow
    [cam2_2x,cam2_2y] = ginput(2)
    matrix2_2(1:2,j) = cam2_2x;
    matrix2_2(3:4,j) = cam2_2y;
end
save('matrix2_2.mat','matrix2_2')
%%
numFrames = size(vidFrames3_2,4);
peak_frame = 79;
matrix2_3 = zeros(4,(numFrames - peak_frame)/2);
for j = 1:numFrames - peak_frame
    X = vidFrames3_2(:,:,:,j*2 + peak_frame - 1);
    imshow(X);
    drawnow
    [cam2_3x,cam2_3y] = ginput(2)
    matrix2_3(1:2,j) = cam2_3x;
    matrix2_3(3:4,j) = cam2_3y;
end
save('matrix2_3.mat','matrix2_3')


%% Case 3
load('cam1_3.mat')
load('cam2_3.mat')
load('cam3_3.mat')
implay(vidFrames1_3)
% Loop to show each frame of the video
numFrames = size(vidFrames1_3,4);
% Found visually
peak_frame = 80;
% numFrames columns starting at peak
matrix3_1 = zeros(2,numFrames - peak_frame);
for j = 1:numFrames - peak_frame
    X = vidFrames1_3(:,:,:,j + peak_frame - 1);
    imshow(X);
    drawnow
    [cam1_3x,cam1_3y] = ginput(1)
    matrix3_1(1,j) = cam1_3x;
    matrix3_1(2,j) = cam1_3y;
end
save('matrix3_1.mat','matrix3_1')
%%
numFrames = size(vidFrames2_3,4);
peak_frame = 66;
```

```
matrix3_2 = zeros(2,numFrames - peak_frame);
for j = 1:numFrames - peak_frame
    X = vidFrames2_3(:,:,:,j + peak_frame - 1);
    imshow(X);
    drawnow
    [cam3_2x,cam3_2y] = ginput(1)
    matrix3_2(1,j) = cam3_2x;
    matrix3_2(2,j) = cam3_2y;
end
save('matrix3_2.mat','matrix3_2')
%%
numFrames = size(vidFrames3_3,4);
peak_frame = 70;
matrix3_3 = zeros(2,numFrames - peak_frame);
for j = 1:numFrames - peak_frame
    X = vidFrames3_3(:,:,:,j + peak_frame - 1);
    imshow(X);
    drawnow
    [cam3_3x,cam3_3y] = ginput(1)
    matrix3_3(1,j) = cam3_3x;
    matrix3_3(2,j) = cam3_3y;
end
save('matrix3_3.mat','matrix3_3')


%% Case 4
load('cam1_4.mat')
load('cam2_4.mat')
load('cam3_4.mat')
implay(vidFrames1_4)
% Loop to show each frame of the video
numFrames = size(vidFrames1_4,4);
% Found visually
peak_frame = 93;
% numFrames columns starting at peak
matrix4_1 = zeros(2,numFrames - peak_frame);
for j = 1:numFrames - peak_frame
    X = vidFrames1_4(:,:,:,j + peak_frame - 1);
    imshow(X);
    drawnow
    [cam1_4x,cam1_4y] = ginput(1)
    matrix4_1(1,j) = cam1_4x;
    matrix4_1(2,j) = cam1_4y;
end
save('matrix4_1.mat','matrix4_1')
%%
```

```
numFrames = size(vidFrames2_4,4);
% Found visually
peak_frame = 99;
% numFrames columns starting at peak
matrix4_2 = zeros(2,numFrames - peak_frame);

for j = 1:numFrames - peak_frame
    X = vidFrames2_4(:,:,:,j + peak_frame - 1);
    imshow(X);
    drawnow
    [cam2_4x,cam2_4y] = ginput(1)
    matrix4_2(1,j) = cam2_4x;
    matrix4_2(2,j) = cam2_4y;
end
save('matrix4_2.mat','matrix4_2')
%%
numFrames = size(vidFrames3_4,4);
% Found visually
peak_frame = 91;
% numFrames columns starting at peak
matrix4_3 = zeros(2,numFrames - peak_frame);

for j = 1:numFrames - peak_frame
    X = vidFrames3_4(:,:,:,j + peak_frame - 1);
    imshow(X);
    drawnow
    [cam3_4x,cam3_4y] = ginput(1)
    matrix4_3(1,j) = cam3_4x;
    matrix4_3(2,j) = cam3_4y;
end
save('matrix4_3.mat','matrix4_3')



%% Case 1 - combine/organize data
matrix1(3:4,:) = matrix1_2(:,1:end-8);
matrix1(5:6,:) = matrix1_3(:,1:end-6);

matrix1(1,:) = matrix1(1,:)/mean(matrix1(1,:));
matrix1(2,:) = matrix1(2,:)/mean(matrix1(2,:));
matrix1(3,:) = matrix1(3,:)/mean(matrix1(3,:));
matrix1(4,:) = matrix1(4,:)/mean(matrix1(4,:));
matrix1(5,:) = matrix1(5,:)/mean(matrix1(5,:));
matrix1(6,:) = matrix1(6,:)/mean(matrix1(6,:));
%% Case 2 - combine/organize data
matrix2_1 = matrix2_1(:,1:120);
```

```
matrix2_2 = matrix2_2(:,1:128);
matrix2_1 = [matrix2_1(1,:)-matrix2_1(2,:);
             matrix2_1(3,:)-matrix2_1(4,:)];
matrix2_2 = [matrix2_2(1,:)-matrix2_2(2,:);
             matrix2_2(3,:)-matrix2_2(4,:)];
matrix2_3 = [matrix2_3(1,:)-matrix2_3(2,:);
             matrix2_3(3,:)-matrix2_3(4,:)];


%%
matrix2 = [matrix2_1(:,1:120);
     matrix2_2(:,1:120);
     matrix2_3(:,1:120)];
matrix2(1,:) = matrix2(1,:)/mean(matrix2(1,:));
matrix2(2,:) = matrix2(2,:)/mean(matrix2(2,:));
matrix2(3,:) = matrix2(3,:)/mean(matrix2(3,:));
matrix2(4,:) = matrix2(4,:)/mean(matrix2(4,:));
matrix2(5,:) = matrix2(5,:)/mean(matrix2(5,:));
matrix2(6,:) = matrix2(6,:)/mean(matrix2(6,:));
%% Case 3 - combine/organize data
matrix3 = [matrix3_1(:,1:159);
     matrix3_2(:,1:159);
     matrix3_3(:,1:159)];
matrix3(1,:) = matrix3(1,:)/mean(matrix3(1,:));
matrix3(2,:) = matrix3(2,:)/mean(matrix3(2,:));
matrix3(3,:) = matrix3(3,:)/mean(matrix3(3,:));
matrix3(4,:) = matrix3(4,:)/mean(matrix3(4,:));
matrix3(5,:) = matrix3(5,:)/mean(matrix3(5,:));
matrix3(6,:) = matrix3(6,:)/mean(matrix3(6,:));
%% Case 4 - combine/organize data
matrix4 = [matrix4_1(:,1:299);
     matrix4_2(:,1:299);
     matrix4_3(:,1:299)];
matrix4(1,:) = matrix4(1,:)/mean(matrix4(1,:));
matrix4(2,:) = matrix4(2,:)/mean(matrix4(2,:));
matrix4(3,:) = matrix4(3,:)/mean(matrix4(3,:));
matrix4(4,:) = matrix4(4,:)/mean(matrix4(4,:));
matrix4(5,:) = matrix4(5,:)/mean(matrix4(5,:));
matrix4(6,:) = matrix4(6,:)/mean(matrix4(6,:));
%% PCA
[U1,S1,V1] = svd(matrix1,'econ');
[U2,S2,V2] = svd(matrix2,'econ');
[U3,S3,V3] = svd(matrix3,'econ');
[U4,S4,V4] = svd(matrix4,'econ');

sig1 = diag(S1);
```

```
sig2 = diag(S2);
sig3 = diag(S3);
sig4 = diag(S4);

energy_case1_1 = sig1(1)^2/sum(sig1.^2);
energy_case1_2 = (sig1(1)^2+sig1(2))/sum(sig1.^2);
energy_case1_3 = (sig1(1)^2+sig1(2)+sig1(3))/sum(sig1.^2);
energy_case1_4 = (sig1(1)^2+sig1(2)+sig1(3)+sig1(4))/sum(sig1.^2);
energy_case1_5 = (sig1(1)^2+sig1(2)+sig1(3)+sig1(4)+sig1(5))/sum(sig1.^2);
energy_case1_6 = (sig1(1)^2+sig1(2)+sig1(3)+sig1(4)+sig1(5)+sig1(6))/sum(sig1.^2

energy_case2_1 = sig2(1)^2/sum(sig2.^2);
energy_case2_2 = (sig2(1)^2+sig2(2))/sum(sig2.^2);
energy_case2_3 = (sig2(1)^2+sig2(2)+sig2(3))/sum(sig2.^2);
energy_case2_4 = (sig2(1)^2+sig2(2)+sig2(3)+sig2(4))/sum(sig2.^2);
energy_case2_5 = (sig2(1)^2+sig2(2)+sig2(3)+sig2(4)+sig2(5))/sum(sig2.^2);
energy_case2_6 = (sig2(1)^2+sig2(2)+sig2(3)+sig2(4)+sig2(5)+sig2(6))/sum(sig2.^2

energy_case3_1 = sig3(1)^2/sum(sig3.^2);
energy_case3_2 = (sig3(1)^2+sig3(2))/sum(sig3.^2);
energy_case3_3 = (sig3(1)^2+sig3(2)+sig3(3))/sum(sig3.^2);
energy_case3_4 = (sig3(1)^2+sig3(2)+sig3(3)+sig3(4))/sum(sig3.^2);
energy_case3_5 = (sig3(1)^2+sig3(2)+sig3(3)+sig3(4)+sig3(5))/sum(sig3.^2);
energy_case3_6 = (sig3(1)^2+sig3(2)+sig3(3)+sig3(4)+sig3(5)+sig3(6))/sum(sig3.^2

energy_case4_1 = sig4(1)^2/sum(sig4.^2);
energy_case4_2 = (sig4(1)^2+sig4(2))/sum(sig4.^2);
energy_case4_3 = (sig4(1)^2+sig4(2)+sig4(3))/sum(sig4.^2);
energy_case4_4 = (sig4(1)^2+sig4(2)+sig4(3)+sig4(4))/sum(sig4.^2);
energy_case4_5 = (sig4(1)^2+sig4(2)+sig4(3)+sig4(4)+sig4(5))/sum(sig4.^2);
energy_case4_6 = (sig4(1)^2+sig4(2)+sig4(3)+sig4(4)+sig4(5)+sig4(6))/sum(sig4.^2



matrix1_rank1 = S1(1,1) * U1(:,1) * V1(:,1)';
figure(1)
plot(matrix1_rank1(1,:),matrix1_rank1(2,:),'r.');

matrix2_rank1 = S2(1,1) * U2(:,1) * V2(:,1)';
figure(2)
plot(matrix2_rank1(1,:),matrix2_rank1(2,:),'r.');

matrix3_rank1 = S3(1,1) * U3(:,1) * V3(:,1)';
figure(3)
plot(matrix3_rank1(1,:),matrix3_rank1(2,:),'r.');

```

```
matrix4_rank1 = S4(1,1) * U4(:,1) * V4(:,1)';
figure(4)
plot(matrix4_rank1(1,:),matrix4_rank1(2,:),'r.');

%%
figure(5)
subplot(2,2,1)
plot(1,1-energy_case1_1,'ko',2,1-energy_case1_2,'ko',3,1-energy_case1_3,'ko',4,1-
title('Case 1 Energies')
xlabel('Dimension (n)')
ylabel('1 - Energy at Dimension (n)')

subplot(2,2,2)
plot(1,1-energy_case2_1,'ko',2,1-energy_case2_2,'ko',3,1-energy_case2_3,'ko',4,1-
title('Case 2 Energies')
xlabel('Dimension (n)')
ylabel('1 - Energy at Dimension (n)')

subplot(2,2,3)
plot(1,1-energy_case3_1,'ko',2,1-energy_case3_2,'ko',3,1-energy_case3_3,'ko',4,1-
title('Case 3 Energies')
xlabel('Dimension (n)')
ylabel('1 - Energy at Dimension (n)')
ylabel('1 - Energy at Dimension (n)')

subplot(2,2,4)
plot(1,1-energy_case4_1,'ko',2,1-energy_case4_2,'ko',3,1-energy_case4_3,'ko',4,1-
title('Case 4 Energies')
xlabel('Dimension (n)')
ylabel('1 - Energy at Dimension (n)')
print(gcf,'energies.png','-dpng')
```