Principle Component Analysis on Videos of Simple Harmonic Oscillators

Jeffrey M Abraham

ABSTRACT

The purpose of this report is to extract the position of a mass from three camera angles for four separate simple harmonic oscillator experiments and perform principle component analysis to describe the oscillatory motion.

Keywords: PCA, Simple Harmonic Oscillator

I INTRODUCTION AND OVERVIEW

This analysis is being done to extract the motion of four separate spring mass system experiments using image analysis methods and PCA. The four experiments consist of a simple case, a case with shaky video footage, a case with pendulum like motion as well as oscillatory motion, and a case with pendulum, oscillatory, and rotational motion. A flashlight was positioned on the top of the mass to make locating it within the image easier, but in the spinning case, as well as some camera angles, the flashlight is not visible the whole time.

II THEORETICAL BACKGROUND

Binary Image Threshold

A binary image threshold takes a grey-scale image and converts all pixel values below a set threshold to zero intensity and all values above such threshold to maximum intensity.

Principle Component Analysis (PCA)

Principle component analysis uses singular value decomposition(SVD) to break a matrix into three matrices, left singular vectors (U), singular values (Σ) and right singular vectors (V). SVD is derived from an eigenvalue decomposition of the input co-variance matrix, but can be done on any size matrix while eigenvalue decomposition can only be done on square matrices. The input matrix can be represented as $U * \Sigma * V^*$ where V^* is the conjugate transpose of V. It can be helpful to have your input matrix have row mean values of 0. Especially for data with little variation relative to non normalized mean. Singular values σ_n represent coefficients for magnitude of variance along left singular vector V_n. Calculations for Energy of rank n approximation in terms of singular values can be seen in (1) where r is the total number of singular values.

$$E_n = \frac{\sum_{i=1}^{n} \sigma_i^2}{\sum_{j=1}^{r} \sigma_j^2} \tag{1}$$

III ALGORITHM IMPLEMENTATION AND DEVELOPMENT

Location Tracking Methodology

I developed a method for tracking the paint can in each video by first using a for loop to look at each frame of the video individually. I developed my method on the first camera angle for the first test and then adapted it for future tests. First I tried converting the frame to gray scale and looking for the maximum intensity position, but some of the background objects had higher intensities than the flashlight, and the flashlight wasn't always pointing towards the camera. This prompted me to look into binary image thresholding and filtering as solutions. Using a filter to black out all parts of the frame where the paint can never swung, the binary image extracted using an intensity threshold of 0.98. The position was then extracted by averaging the x and y positions of all non-zero values. An example of one binary image and

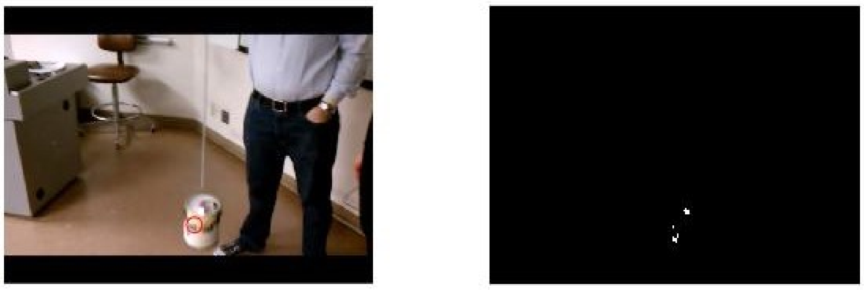

Figure 1. Example Frame With Plotted Position (left) and Binary Frame (right)

its extracted position can be seen in Figure 1. For other camera angles, filter location and threshold value were adjusted to get similar results. The third camera angle was taken landscape, so I just swapped x and y coordinates to account for this.

Aligning Motion Across Camera Angles

Since each video was not started at the same instant, and is not the same length some work was needed to align the motion of the paint can across videos and get vectors of the same length for use in analysis of the motion later. I accomplished this by plotting the y positions recorded for each camera angle and recorded the frame in which the paint can hit its first local maximum. I then threw out all of the data before these points and shortened the longer two videos to be the same length as the shortest one. An example can be seen in Figure 2.

Dimensionality Analysis

For camera angles a, b, and c one matrix A was assembled as seen in (2). Singular value decomposition was then preformed and the energy represented by each rank approximation calculated.

$$\begin{vmatrix} -x_a- \\ -y_a- \\ -x_b- \\ -y_b- \\ -x_c- \\ -y_c- \end{vmatrix} \tag{2}$$

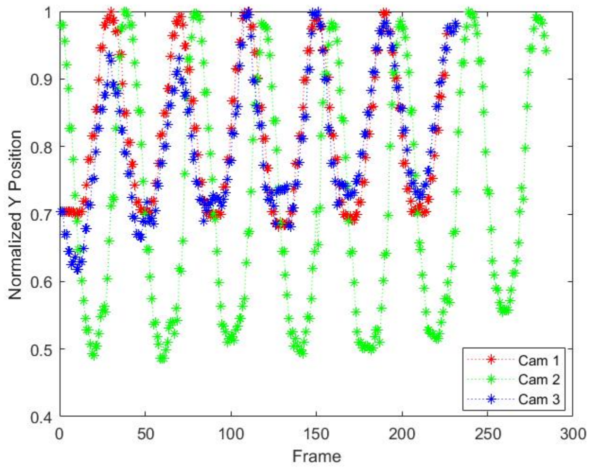

Figure 2. Plot Used for Aligning Camera Angles in Time

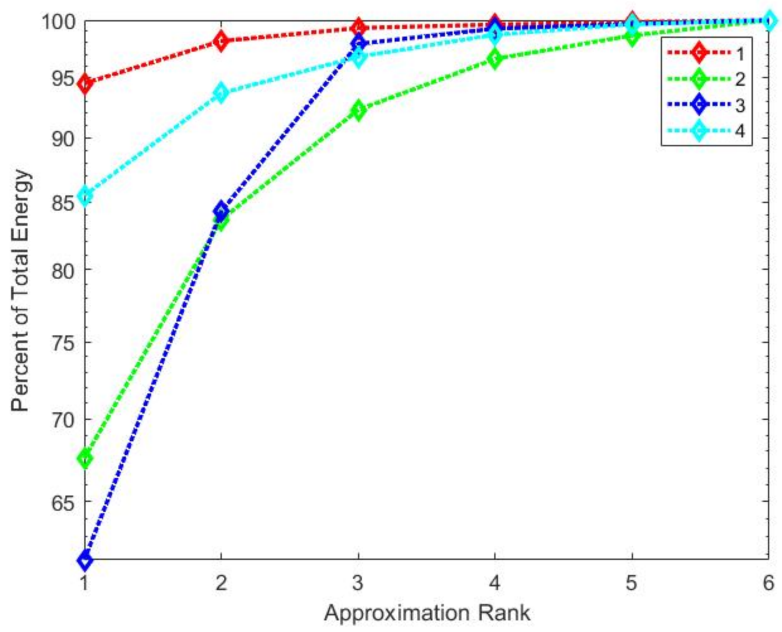

Figure 3. Rank Energies

IV COMPUTATIONAL RESULTS

Plotting the energies per rank approximation for each test as seen in Figure 3, an analysis of the dimensionality of the motion in each test can be done. The first test contained 94.5% of the total energy in its first rank approximation which represents 1 dimensional motion. The shaky test had the most energy represented in the higher rank approximations. This can be explained by the inconsistent movement caused by the camera shake impacting all six dimensions. The third test had more than 95% of their energy represented in the third rank approximation. This means that the motion can be mostly constrained to the basis of the first three basis vectors. Plotting the first two left singular vectors for each camera in the third test, the paint cans motion is clearly described. These plots can be seen in Figure 4. The fourth test was more rank one than test three as evident from the energy contained in the first rank approximation. This makes sense because the swinging in the fourth test was less pronounced than in test three.

V SUMMARY AND CONCLUSIONS

Through principle component analysis I was able to describe the motion of 4 spring mass systems using data gathered from three camera angles. This was used to do analysis of the dimension of motion in each test. I also discovered that shaky footage adds random motion to all directions and increases the high rank energies.

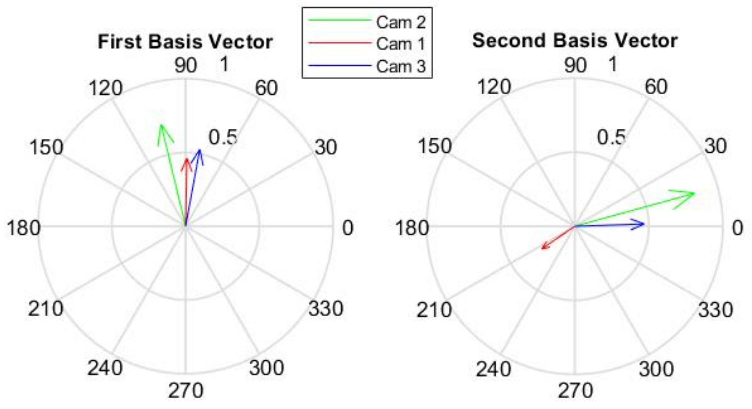

Figure 4. Test Three Basis Vectors

.

APPENDIX A.

Matlab Functions

imbinarize(I,value)
replaces all pixel intensities below a set value with 0 and above with 1

im2double(I)
Converts from an int8 image format to double precision rescaled between 0 and 1.

find(X)
Returns the indices of all non-zero values within matrix X

ind2sub(sz, ind)
Converts linear index into row and column indices with specified size

APPENDIX B.

Test 1

```
clc; clear all; close all;

load('cam1_1.mat')
fa = size(vidFrames1_1,4);
xa = zeros(1,226);
ya = zeros(1,226);
filtera = zeros(480,640);
filtera(:,280:400)=ones(480,121);
filterc = zeros(480,640);
filterc(:,260:640)=ones(480,381);
for j = 1:fa
    frame = vidFrames1_1(:,:,:,j);
    frame = imbinarize(filtera.*im2double(rgb2gray(frame)),0.98);
    [yind,xind] = ind2sub([480,640],find(frame));
    xa(j) = round(mean(xind));
    ya(j) = round(mean(yind));
end
%%

load('cam2_1.mat')
fb = size(vidFrames2_1,4);
xb = zeros(1,fb);
yb = zeros(1,fb);

for k = 1:fb
    frame = vidFrames2_1(:,:,:,k);
    frame = imbinarize(im2double(rgb2gray(frame)),0.98);
    [yind,xind] = ind2sub([480 640],find(frame));
    xb(k) = round(mean(xind));
    yb(k) = round(mean(yind));
end

%%

load('cam3_1.mat')
fc = size(vidFrames3_1,4);
xc = zeros(1,fc);
yc = zeros(1,fc);

for l = 1:fc
    frame = vidFrames3_1(:,:,:,l);
    frame = imbinarize(filterc.*im2double(rgb2gray(frame)),0.96);
    [xind,yind] = ind2sub([480 640],find(frame));
    yc(l) = round(mean(yind));
    xc(l) = round(mean(xind));
end
%%

plot(ya/max(ya),'r*:')
%30
hold on;
plot(yb/max(yb),'g*:')
```

```
%39
plot(yc/max(yc),'b*:')
%29
legend('Cam 1', 'Cam 2', 'Cam 3')
xlabel('Frame')
ylabel('Normalized Y Position')

%%
for m = 1:length(xa)-35
    plot(xa(m+30:m+35),ya(m+30:m+35),'r*:')
    hold on;
    plot(xb(m+39:m+44),yb(m+39:m+44),'g*:')
    plot(xc(m+29:m+34),yc(m+29:m+34),'b*:');drawnow;
    hold off;
    axis([0 480 0 640])
    pause(0.05)
end
%%
xa = xa(30:end); ya = ya(30:end);
xb = xb(39:length(xa)+38); yb = yb(39:length(xa)+38);
xc = xc(29:length(xa)+28); yc = yc(29:length(xa)+28);
A = [xa; ya; xb; yb; xc; yc];
A = A - mean(A,2);
[U,S,V] = svd(A);
A1 = S(1,1)*U(:,1)*V(:,1)';
%%
plot(A1(2,:),'r*:')
hold on;
plot(A(2,:),'b.-')
plot(A(4,:),'g.-')
plot(A(6,:),'k.-')
axis([0 197 -100 100])
ylabel('Y position'); xlabel('time (frames)')
legend('Rank 1','Cam 1','Cam 2', 'Cam 3')
%%
energy1 = [S(1,1)^2 S(2,2)^2 S(3,3)^2 S(4,4)^2 S(5,5)^2 S(6,6)^2]./
    sum(sum(S.^2));
save('energy1.mat','energy1');
%%

semilogy([S(1,1) S(2,2) S(3,3) S(4,4) S(5,5) S(6,6)],'rd')
```

Test 2

```
clc; clear all; close all;

load('cam1_2.mat')
fa = size(vidFrames1_2,4);
xa = zeros(1,fa);
ya = zeros(1,fa);
filtera = zeros(480,640);
filtera(:,320:640) = ones(480,321);
filteras = zeros(480,640);
filteras(1:240,1:320) = ones(240,320);
filterc = zeros(480,640);
filterc(:,240:640)=ones(480,401);
for k = 1:fa
```

```
    frame = vidFrames1_2(:,:,:,k);
    frame = imbinarize(filtera.*im2double(rgb2gray(frame)),0.98);
    [yind,xind] = ind2sub([480,640],find(frame));
    xa(k) = round(mean(xind));
    ya(k) = round(mean(yind));
end

%%

load('cam2_2.mat')
fb = size(vidFrames2_2,4);
xb = zeros(1,fb);
yb = zeros(1,fb);

for j = 1:fb
    frame = vidFrames2_2(:,:,:,j);
    frame = imbinarize(im2double(rgb2gray(frame)),0.98);
    [yind,xind] = ind2sub([480 640],find(frame));
    xb(j) = round(mean(xind));
    yb(j) = round(mean(yind));
end

%%

load('cam3_2.mat')
fc = size(vidFrames3_2,4);
xc = zeros(1,fc);
yc = zeros(1,fc);

for l = 1:fc
    frame = vidFrames3_2(:,:,:,l);
    frame = imbinarize(filterc.*im2double(rgb2gray(frame)),0.96);
    [xind,yind] = ind2sub([480 640],find(frame));
    xc(l) = round(mean(xind));
    yc(l) = round(mean(yind));
end
%%

plot(ya,'r*')
hold on;
plot(yb,'g*')
plot(yc,'b*')

%%
xa = xa(12:end); ya = ya(12:end);
xb = xb(1:length(xa)); yb = yb(1:length(xa));
xc = xc(16:length(xa)+15); yc = yc(16:length(xa)+15);

%%
A = [xa; ya; xb; yb; xc; yc];
A = A - mean(A,2);
AS = [smooth(xa)';smooth(ya)';smooth(xb)';smooth(yb)';smooth(xc)';
    smooth(yc)'];
AS = AS - mean(AS,2);
[U,S,V] = svd(A);
```

```
[U2,S2,V2] = svd(AS);
A1 = S(1,1)*U(:,1)*V(:,1)';
energy1 = S(1,1)^2/sum(sum(S.^2));
energy2 = S(2,2)^2/sum(sum(S.^2));
%%

plot(A1(1,:),'r*:')
hold on;
plot(A(2,:),'b.-')
plot(A(4,:),'g.-')
plot(A(6,:),'k.-')
axis([0 303 0 640])
ylabel('Y position'); xlabel('time (frames)')
legend('Rank 1','Cam 1','Cam 2', 'Cam 3')
%%
for l = 1:length(xa)-5
    plot(xa(l:l+5),ya(l:l+5),'r*:');
    hold on;
    plot(xb(l:l+5),yb(l:l+5),'g*:');
    plot(xc(l:l+5),yc(l:l+5),'b*:');drawnow;
    hold off;
    axis([0 480 0 640])
    pause(0.05)
end
%%
energy2 = [S(1,1)^2 S(2,2)^2 S(3,3)^2 S(4,4)^2 S(5,5)^2 S(6,6)^2]./
    sum(sum(S.^2));
energy2s = [S2(1,1)^2 S2(2,2)^2 S2(3,3)^2 S2(4,4)^2 S2(5,5)^2 S2(6,6)
    ^2]./sum(sum(S2.^2));
save('energy2.mat','energy2');

semilogy(cumsum(energy2),'gd:','Linewidth',2)
hold on;
semilogy(cumsum(energy2s),'yd:','Linewidth',2)
xticks([1 2 3 4 5 6])
legend('Raw Data','With Smothing Function')
ylabel('Percent of Total Energy')
xlabel('Approximation Rank')
```

Test 3

```
clc; clear all; close all;

load('cam1_3.mat')
fa = size(vidFrames1_3,4);
xa = zeros(1,fa);
ya = zeros(1,fa);
filtera = zeros(480,640);
filtera(121:480,280:480) = ones(360,201);
filterc = zeros(480,640);
filterc(:,240:640)=ones(480,401);
for k = 1:fa
    frame = vidFrames1_3(:,:,:,k);
    frame = imbinarize(filtera.*im2double(rgb2gray(frame)),0.98);
    [yind,xind] = ind2sub([480,640],find(frame));
    xa(k) = round(mean(xind));
    ya(k) = round(mean(yind));
```

```
end

%%

load('cam2_3.mat')
fb = size(vidFrames2_3,4);
xb = zeros(1,fb);
yb = zeros(1,fb);

for j = 1:fb
    frame = vidFrames2_3(:,:,:,j);
    frame = imbinarize(im2double(rgb2gray(frame)),0.98);
    [yind,xind] = ind2sub([480 640],find(frame));
    xb(j) = round(mean(xind));
    yb(j) = round(mean(yind));
end

%%

load('cam3_3.mat')
fc = size(vidFrames3_3,4);
xc = zeros(1,fc);
yc = zeros(1,fc);

for l = 1:fc
    frame = vidFrames3_3(:,:,:,l);
    frame = imbinarize(filterc.*im2double(rgb2gray(frame)),0.96);
    [xind,yind] = ind2sub([480 640],find(frame));
    xc(l) = round(mean(xind));
    yc(l) = round(mean(yind));
end
%%

plot(ya,'r*')
hold on;
plot(yb,'g*')
plot(yc,'b*')

%%
xa = xa(14:end); ya = ya(14:end);
xc = xc(8:length(xa)+7); yc = yc(8:length(xa)+7);
xb = xb(1:length(xa)); yb = yb(1:length(xa));

%%
A = [xa; ya; xb; yb; xc; yc];
A = A - mean(A,2);
[U,S,V] = svd(A);
A1 = S(1,1)*U(:,1)*V(:,1)';
energy1 = S(1,1)^2/sum(sum(S.^2));
energy2 = S(2,2)^2/sum(sum(S.^2));
%%

plot(A1(1,:),'r*:')
hold on;
plot(A(2,:),'b.-')
plot(A(4,:),'g.-')
plot(A(6,:),'k.-')
axis([0 303 0 640])
ylabel('Y position'); xlabel('time (frames)')
title(['Rank 1 Approximation Preserves ' string(100*energy1) 'Percent
    of Energy'])
legend('Rank 1','Cam 1','Cam 2', 'Cam 3')
%%
for l = 1:length(xc)-5
    plot(xa(l:l+5),ya(l:l+5),'r*:');
    hold on;
    plot(xb(l:l+5),yb(l:l+5),'g*:');
    plot(xc(l:l+5),yc(l:l+5),'b*:');drawnow;
    hold off;
    axis([0 480 0 640])
    pause(0.05)
end
%%
energy3 = [S(1,1)^2 S(2,2)^2 S(3,3)^2 S(4,4)^2 S(5,5)^2 S(6,6)^2]./
    sum(sum(S.^2));
save('energy3.mat','energy3');

%%
semilogy([S(1,1) S(2,2) S(3,3) S(4,4) S(5,5) S(6,6)],'rd')

%%
subplot(1,2,1)
compass(U(3,1),U(4,1),'g')
hold on;
compass(U(1,1),U(2,1),'r')
compass(U(5,1),U(6,1),'b')
title('First Basis Vector')
subplot(1,2,2)
compass(U(3,2),U(4,2),'g')
hold on;
compass(U(1,2),U(2,2),'r')
compass(U(5,2),U(6,2),'b')
title('Second Basis Vector')
legend('Cam 2', 'Cam 1', 'Cam 3')
```

Test 4

```
clc; clear all; close all;

load('cam1_4.mat')
fa = size(vidFrames1_4,4);
xa = zeros(1,fa);
ya = zeros(1,fa);
filtera = zeros(480,640);
filtera(121:480,280:480) = ones(360,201);
filterc = zeros(480,640);
filterc(1:320,240:640)=ones(320,401);
for k = 1:fa
    frame = vidFrames1_4(:,:,:,k);
    frame = imbinarize(filtera.*im2double(rgb2gray(frame)),0.97);
    [yind,xind] = ind2sub([480,640],find(frame));
    xa(k) = round(mean(xind));
    ya(k) = round(mean(yind));
end

%%

load('cam2_4.mat')
fb = size(vidFrames2_4,4);
xb = zeros(1,fb);
yb = zeros(1,fb);

for j = 1:fb
    frame = vidFrames2_4(:,:,:,j);
    frame = imbinarize(im2double(rgb2gray(frame)),0.98);
    [yind,xind] = ind2sub([480 640],find(frame));
    xb(j) = round(mean(xind));
    yb(j) = round(mean(yind));
end

%%

load('cam3_4.mat')
fc = size(vidFrames3_4,4);
xc = zeros(1,fc);
yc = zeros(1,fc);

for l = 1:fc
    frame = vidFrames3_4(:,:,:,l);
    imshow(frame)
    frame = imbinarize(filterc.*im2double(rgb2gray(frame)),0.93);
    [xind,yind] = ind2sub([480 640],find(frame));
    xc(l) = round(mean(xind));
    yc(l) = round(mean(yind));
    hold on;
    plot(yc(l),xc(l),'ro');drawnow;
    hold off;
end
%%

plot(ya,'r*')
hold on;
plot(yb,'g*')
plot(yc,'b*')

%%
xa = xa(18:end); ya = ya(18:end);
xc = xc(19:length(xa)+18); yc = yc(19:length(xa)+18);
xb = xb(25:length(xa)+24); yb = yb(25:length(xa)+24);

%%
A = [xa; ya; xb; yb; xc; yc];
A = A - mean(A,2);
[U,S,V] = svd(A);
A1 = S(1,1)*U(:,1)*V(:,1)';
energy1 = S(1,1)^2/sum(sum(S.^2));
energy2 = S(2,2)^2/sum(sum(S.^2));
```

```
%%

plot(A1(1,:),'r*:')
hold on;
plot(A(2,:),'b.-')
plot(A(4,:),'g.-')
plot(A(6,:),'k.-')
axis([0 303 0 640])
ylabel('Y position'); xlabel('time (frames)')
title(['Rank 1 Approximation Preserves ' string(100*energy1) 'Percent
    of Energy'])
legend('Rank 1','Cam 1','Cam 2', 'Cam 3')
%%
for l = 1:length(xc)-5
    plot(xa(l:l+5),ya(l:l+5),'r*:');
    hold on;
    plot(xb(l:l+5),yb(l:l+5),'g*:');
    plot(xc(l:l+5),yc(l:l+5),'b*:');drawnow;
    hold off;
    axis([0 480 0 640])
    pause(0.05)
end
%%
energy4 = [S(1,1)^2 S(2,2)^2 S(3,3)^2 S(4,4)^2 S(5,5)^2 S(6,6)^2]./
   sum(sum(S.^2));
save('energy4.mat','energy4');
```