AMATH 482: Principal Component Analysis on Paint Can Action (PCA on PCA)

Anh-Minh Nguyen

February 9, 2020

Abstract In this report, there are a total of 3 cameras recording a paint can in motion. There are four different cases. Our goal will be to extract information about the mass positions of the paint can then perform PCA.

1 Introduction and Overview

To begin, we have 4 different scenarios on which we will be performing Principal Component Analysis (PCA) on. There will be 3 cameras recording the paint can, in the same position, for reach scenario. For this report, we will only track the position of the can itself in the context of the x-y plane.

- Test 1: The paint can just bounces up and down happily in the z-direction, no camera shake is involved. We can clearly see the flashlight attached to the top of the can.
- Test 2: This is the same as Test 1 except the camera people are shaking the camera to introduce noise. The flashlight on top can still be seen.
- Test 3: The paint can is released off-center so there is motion in the x-y plane as well. We now have a pendulum motion and simple harmonic oscillations. The flashlight is harder to spot
- Test 4: This is like Test 3, but the can is rotating as well. The flashlight is hidden in some frames in this test.

As we can see, the challenge will lie in extracting the positions of the paint can from the videos before we can even begin Principal Component Analysis.

2 Theoretical Background

To begin, we want to discuss Singular Value Decomposition (SVD). Consider a matrix A, which has the m x n dimensions. Now, we consider the matrices AA^T and A^TA, which are both square, have the same rank as A, have positive eigenvalues and other properties. We can call the basis of eigenvectors of AA^T the matrix U, and the basis of eigenvectors of A^TA the matrix V. Let us also define a diagonal matrix S, where the values on the diagonal are the square roots of the eigenvalues, ordered from largest to smallest. SVD states that we can any matrix A as

$$A = USV^T \tag{1}$$

Note that U and V are orthogonal matrices. Moving on, let's say we have a data matrix, X, where the rows contain information about the observations, defined by the columns. Now, let's say we subtract the row mean, μ, from each row. Then, we can calculate the variance of each row as

$$\sigma^2 = \frac{1}{n}\vec{a}\vec{a}^T \tag{2}$$

where n is the number of columns and a is the particular row of data. We want to calculate the covariance matrix, C_x:

$$C_x = \frac{1}{n-1} X X^T \tag{3}$$

Large variances pertain to large dynamics of interest, and low variances relate to non-interesting dynamics. The covariance matrix tells us information about the redundancy and finding signals with the highest variance. We want to diagonalize C_x to remove redundancies and leave behind the Principal components. Thus, we can use SVD to decompose the matrix and conduct principal component analysis.
We can standardize the matrix by multiplying each element by $\frac{1}{\sqrt{(n-1)}}$ then use SVD. By multiplying the transpose of the matrix U we get from SVD with our data matrix X, we have the diagonal matrix, Y.

$$Y = U' * X \rightarrow C_Y = \frac{1}{n-1}\sigma^2 \tag{4}$$

And Y is the collection of our principal components projections.

3 Algorithm Implementation and Development

However, our challenge beings in extracting the information from the video. We will use code from the course notes to extract each frame and converting them to images. One thing that is placed in the video to help us is the flashlight, often the brightest "image" in the video. The key then is to identify the "brightest" or "whitest" object. We can help ourselves by converting each image from RGB to gray, technically RGB still, to have an easier time identifying the flashlight. For the last two tests, we can use the white of the paint can to track it. This works because RGB for the color white is 255-255-255 and in gray scale, it is white. So, if we search for the max of each frame, theoretically, it should be the paint can. However, the paint can is not the only white colored "thing", so we want to grab more than just the max grayscale pixel. We will grab a couple depending on whether they are close enough to the max grayscale value. Next, we can find their values and average the indices, and it should be the paint can/flashlight.
We can also help ourselves by manually, but painfully, cropping out portions of the frames before searching for the maximum RGB value. This limits the white of other things in the frame like the shoe and the wall. This will allow us to grab the mass positions. Because the 2nd test is much noisier, we will take multiple measurements and average them in an attempt to "average out" some noise.
After, recall that we have 3 cameras, so we manually cut some frames off the beginning and end to match the lengths. then, we will perform Fourier Transforms and apply a Gaussian Filter to remove some noise and then inverse Fourier transform. We create our matrix after that, subtracting the mean and such. Finally, we can create our diagonal matrix, Y and plot the principal components.

4 Computational Results

To analyze our results, we will look at each test and their computational results. For the mass positions, the first row is the camera of the "initial" position, the second row is a lower angle view from camera two, and the third row is the camera that's tilted sideways.

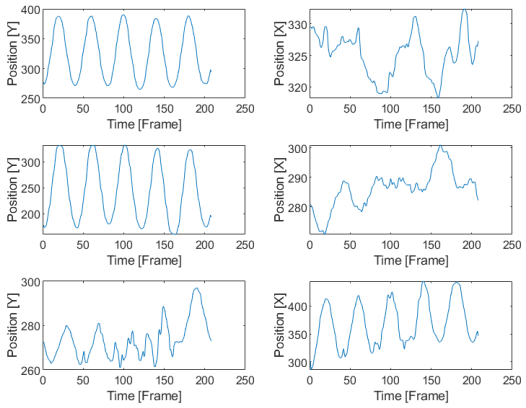

Figure 1: Mass Position Test 1

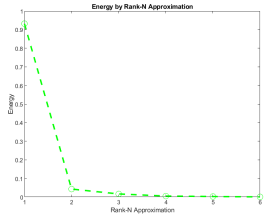

Figure 2: Energy Test 1

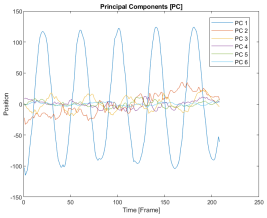

Figure 3: Principal Components

For test 1, we see that the first Principal component tells us the most information, and much of the other information is redundant. The energy gained from including the other components is almost unnecessary. We only need the first component to represent the original positions of the mass.

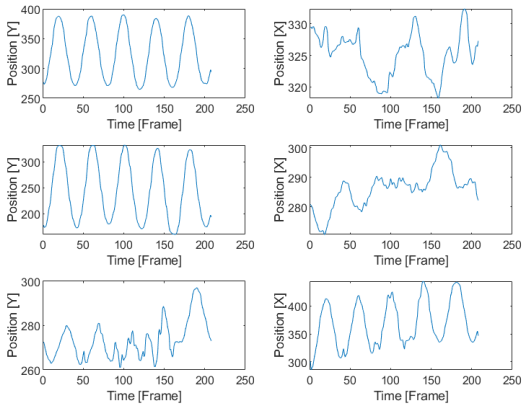

Figure 4: Mass Position Test 2

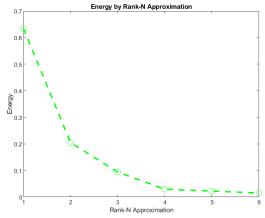

Figure 5: Energy Test 2

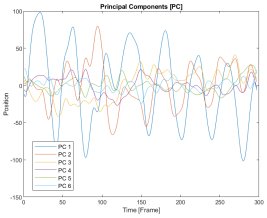

Figure 6: Principal Components

The second test features noise, and a lot of it. Even though the mass is moving in only one direction, the camera shaking brings in a lot of error. The second component is much larger than it should be, as the rest are, and the first component has a much lower energy. We should only need the first component, but noise forces corrupts the data. The energy is more steadily decreasing cumulatively.

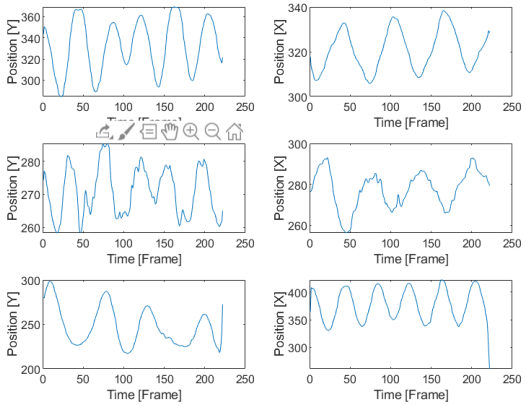

Figure 7: Mass Position Test 3

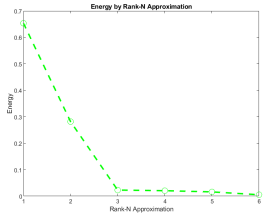

Figure 8: Energy Test 3

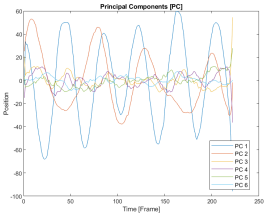

Figure 9: Principal Components

This is the first test where we introduce the pendulum motion in the x-y plane. We see that the first two Principal components are essential. And most of the energy is within the first two modes. The other components may be redundant.

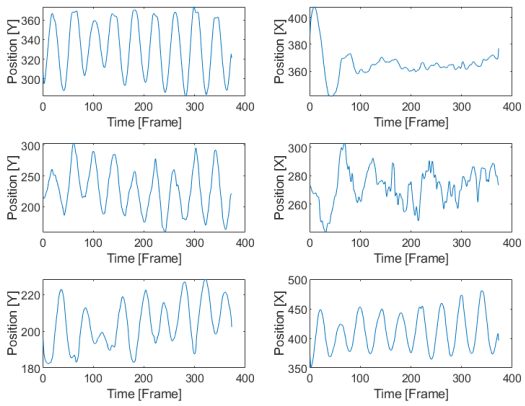

Figure 10: Mass Position Test 4

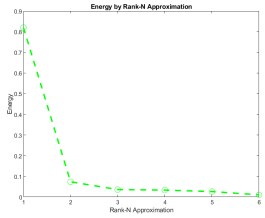

Figure 11: Energy Test 4

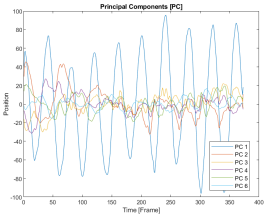

Figure 12: Principal Components

I believe this test was suppose to have a pendulum motion. We can see from the second component that it indicates that, but if falls off. The first component is still has the most energy and tells us enough about the original signal. The rotation should not have affected the mass positions and from our Principal components, it did not seem to.

5 Summary and Conclusions

We see for most of the cases, the first rank approximations seem to tell us or should tell us enough about the original signal. For the third case, the 2nd rank approximation was enough. The PCA (and SVD) does an excellent job at minimizing redundancies. Combined with some Fourier Transforming and filtering,

6 Appendix

A MATLAB Functions

- *frame2im*: Takes a movie frame and gives the RGB image
- *rgb2gray*: Converts RGB image to grayscale, (reduces the 3 dimensions of RGB to 1 dimension of grayscale)
- *fft*: Fourier Transform of an 1-dimensional
- *ifft*: Inverse Fourier Transform of a 1-dimensional object
- *svd*: Decomposes a matrix through singular value decomposition

B MATLAB Code

```
clear; close all; clc;
load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')
%%
numFrames1 = size(vidFrames1_1,4);
%% Test 1
for k = 1 : numFrames1
    mov(k).cdata = vidFrames1_1(:,:,:,k);
    mov(k).colormap = [];
    mov2(k).cdata = vidFrames2_1(:,:,:,k);
    mov2(k).colormap = [];
    mov3(k).cdata = vidFrames3_1(:,:,:,k);
    mov3(k).colormap = [];
end

x1 = zeros(1, numFrames1); y1 = x1;
x11 = zeros(1, numFrames1); y11 = x11;
x12 = zeros(1, numFrames1); y12 = x12;
for jj = 1:numFrames1
   X = rgb2gray(frame2im(mov(jj)));
   X(:, [1:300 390:end]) = 0; % hand calculated
   X(1:200, :) = 0; % hand calculated
   [mmY, mmX] = find(X > max(X(:)) - 10);
   y1(jj) = mean(mmY);
   x1(jj) = mean(mmX);
end


for jj = 1:numFrames1
   X = rgb2gray(frame2im(mov2(jj)));
   X(:, [1:250 365:end]) = 0; % hand calculated
   X([1:85 349:end], :) = 0; % hand calculated
   [mmY, mmX] = find(X > max(X(:)) - 10);
   y11(jj) = mean(mmY);
   x11(jj) = mean(mmX);
end


for jj = 1:numFrames1
   X = rgb2gray(frame2im(mov3(jj)));
   X(:, [1:262 453:end]) = 0; % hand calculated
   X([1:236 334:end], :) = 0; % hand calculated
   [mmY, mmX] = find(X > max(X(:)) - 10); % gives the horizontal and vertical locations
   y12(jj) = mean(mmY);
   x12(jj) = mean(mmX);
end

%% Truncating
[~, ind] = min(y1(1:25));
x1 = x1(ind:end);
y1 = y1(ind:end);
```

```
[~, ind2] = min(y11(1:30));
x11 = x11(ind2:end);
y11 = y11(ind2:end);
[~, ind3] = min(x12(1:20));
x12 = x12(ind3:end);
yl2 = y12(ind3:end);
%%
sh = min([length(x1), length(x11), length(x12)]);
x1 = x1(1:end - (length(x1) - sh));  y1 = y1(1:end - (length(y1) - sh));
x11 = x11(1:end - (length(x11) - sh)); y11 = y11(1:end - (length(y11) - sh));
x12 = x12(1:end - (length(x12) - sh)); y12 = y12(1:end - (length(y12) - sh));

%% FFT for smoothing
n = length(y1);
L = length(y1);
k = (2*pi/L)*[0:n/2-1 -n/2:-1];
tau = .2;
k0 = 0;
filter = exp(-tau*(k-k0).^2);
y1 = abs(ifft(filter.*fft(y1)));
y11 = abs(ifft(filter.*fft(y11)));
y12 = abs(ifft(filter.*fft(y12)));
x1 = abs(ifft(filter.*fft(x1)));
x11 = abs(ifft(filter.*fft(x11)));
x12 = abs(ifft(filter.*fft(x12)));

%%
figure(1)
subplot(3,2,1)
plot(y1)
xlabel('Time [Frame]')
ylabel('Position [Y]')
set(gca,'Fontsize',8)
subplot(3,2,2)
plot(x1)
ylabel('Position [X]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
subplot(3,2,3)
plot(y11)
ylabel('Position [Y]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
subplot(3,2,4)
plot( x11)
ylabel('Position [X]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
subplot(3,2,5)
plot(y12)
ylabel('Position [Y]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
subplot(3,2,6)
```

```
plot(x12)
ylabel('Position [X]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)



%% SVD
XMAT = [x1; y1; x11; y11; x12; y12];
[m,n]=size(XMAT);   %  compute data size
mn=mean(XMAT,2); %  compute mean for each row
XMAT=XMAT-repmat(mn,1,n);
X = XMAT;

[U,S,V]=svd(X/sqrt(n-1));  % perform the SVD
lambda=diag(S).^2;  % produce diagonal variances
Y = U'*X;% produce the principal components projection
%%
figure(2)
plot(lambda./sum(lambda), 'go', 'MarkerSize', 8);
hold on;
plot(lambda./sum(lambda), 'g--', 'Linewidth',2);
hold off;
set(gca, 'Fontsize', 8, 'Xtick',0:6)
title('Energy by Rank-N Approximation')
xlabel('Rank-N Approximation')
ylabel('Energy')

figure(3)
plot(Y(1,:)); hold on;
plot(Y(2,:));
plot(Y(3,:));
plot(Y(4,:));
plot(Y(5,:));
plot(Y(6,:)); hold off;
set(gca, 'Fontsize', 8)
title('Principal Components [PC]')
legend('PC 1', 'PC 2', 'PC 3', 'PC 4', 'PC 5', ...
    'PC 6', 'Location', 'best')
xlabel('Time [Frame]')
ylabel('Position')

%%%%%%% PART 2 %%%%%%%%%%%%
close all; clc; clear;
load('cam1_2.mat')
load('cam2_2.mat')
load('cam3_2.mat')
numFrames2 = size(vidFrames1_2,4);

%% Test 2
for k = 1 : numFrames2
    mov(k).cdata = vidFrames1_2(:,:,:,k);
    mov(k).colormap = [];
```

```
    mov2(k).cdata = vidFrames2_2(:,:,:,k);
    mov2(k).colormap = [];
    mov3(k).cdata = vidFrames3_2(:,:,:,k);
    mov3(k).colormap = [];
end

x2 = zeros(1, numFrames2); y2 = x2;
slider = [15 18 19 20 22 25 30];
for kk = 1:length(slider)
    x24 = zeros(1, numFrames2); y24 = x24;
    for jj = 1:numFrames2
        X = rgb2gray(frame2im(mov(jj)));
        X(:, [1:256 441:end]) = 0; % hand calculated X
        X([1:193 292:end], :) = 0; % hand calculated Y
        [mmY, mmX] = find(X > max(X(:)) - slider(kk));
        y24(jj) = mean(mmY);
        x24(jj) = mean(mmX);
    end
    y2 = y2 + y24;
    x2 = x2 + x24;
end
y2 = y2 ./ length(slider);
x2 = x2 ./ length(slider);

x21 = zeros(1, numFrames2); y21 = x21;
for kk = 1:length(slider)
    x24 = zeros(1, numFrames2); y24 = x24;
    for jj = 1:numFrames2
        X = rgb2gray(frame2im(mov2(jj)));
        X(:, [1:248 481:end]) = 0; % hand calculated X
        X([1:76 362:end], :) = 0; % hand calculated Y
        [mmY, mmX] = find(X > max(X(:)) - slider(kk));
        y24(jj) = mean(mmY);
        x24(jj) = mean(mmX);
    end
    y21 = y21 + y24;
    x21 = x21 + x24;
end

y21 = y21 ./ length(slider);
x21 = x21 ./ length(slider);

x22 = zeros(1, numFrames2); y22 = x22;
for kk = 1:length(slider)
    x24 = zeros(1, numFrames2); y24 = x24;
    for jj = 1:numFrames2
        X = rgb2gray(frame2im(mov3(jj)));
        X(:, [1:270 452:end]) = 0; % hand calculated X
        X([1:172 312:end], :) = 0; % hand calculated Y
        [mmY, mmX] = find(X > max(X(:)) - slider(kk));
        y24(jj) = mean(mmY);
        x24(jj) = mean(mmX);
    end
    y22 = y22 + y24;
```

```
    x22 = x22 + x24;
end
y22 = y22 ./ length(slider);
x22 = x22 ./ length(slider);

[~, ind] = min(y2(1:50));
x2 = x2(ind:end);
y2 = y2(ind:end);
[~, ind2] = min(y21(1:50));
x21 = x21(ind2:end);
y21 = y21(ind2:end);
[~, ind3] = min(x22(1:10));
x22 = x22(ind3:end);
y22 = y22(ind3:end);
y1 = y2; y11 = y21; y12 = y22;
x1 = x2; x11 = x21; x12 = x22;

%%
sh = min([length(x1), length(x11), length(x12)]);
x1 = x1(1:end - (length(x1) - sh));  y1 = y1(1:end - (length(y1) - sh));
x11 = x11(1:end - (length(x11) - sh)); y11 = y11(1:end - (length(y11) - sh));
x12 = x12(1:end - (length(x12) - sh)); y12 = y12(1:end - (length(y12) - sh));

%% FFT for smoothing
n = length(y1);
L = length(y1);
k = (2*pi/L)*[0:(n-1)/2 -(n-1)/2:-1];
tau = 5;
k0 = 0;
filter = exp(-tau*(k-k0).^2);
y1 = abs(ifft(filter.*fft(y1)));
y11 = abs(ifft(filter.*fft(y11)));
y12 = abs(ifft(filter.*fft(y12)));
x1 = abs(ifft(filter.*fft(x1)));
x11 = abs(ifft(filter.*fft(x11)));
x12 = abs(ifft(filter.*fft(x12)));

%%
figure(1)
subplot(3,2,1)
plot(y1)
xlabel('Time [Frame]')
ylabel('Position [Y]')
set(gca,'Fontsize',8)
subplot(3,2,2)
plot(x1)
ylabel('Position [X]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
subplot(3,2,3)
plot(y11)
ylabel('Position [Y]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
```

```
subplot(3,2,4)
plot( x11)
ylabel('Position [X]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
subplot(3,2,5)
plot(y12)
ylabel('Position [Y]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
subplot(3,2,6)
plot(x12)
ylabel('Position [X]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
%% SVD
XMAT = [x1; y1; x11; y11; x12; y12];
[m,n]=size(XMAT);   %  compute data size
mn=mean(XMAT,2); %  compute mean for each row
XMAT=XMAT-repmat(mn,1,n);
X = XMAT;

[U,S,V]=svd(X/sqrt(n-1));  % perform the SVD
lambda=diag(S).^2;  % produce diagonal variances
Y = U'*X;% produce the principal components projection
%%
figure(2)
plot(lambda./sum(lambda), 'go', 'MarkerSize', 8);
hold on;
plot(lambda./sum(lambda), 'g--', 'Linewidth',2);
hold off;
set(gca, 'Fontsize', 8, 'Xtick',0:6)
title('Energy by Rank-N Approximation')
xlabel('Rank-N Approximation')
ylabel('Energy')

figure(3)
plot(Y(1,:)); hold on;
plot(Y(2,:));
plot(Y(3,:));
plot(Y(4,:));
plot(Y(5,:));
plot(Y(6,:)); hold off;
set(gca, 'Fontsize', 8)
title('Principal Components [PC]')
legend('PC 1', 'PC 2', 'PC 3', 'PC 4', 'PC 5', ...
    'PC 6', 'Location', 'best')
xlabel('Time [Frame]')
ylabel('Position')

%%%%%%%% PART 3 %%%%%%%%%%%%%%%%
clear; clc; close all;
load('cam1_3.mat')
load('cam2_3.mat')
```

```
load('cam3_3.mat')

numFrames3 = size(vidFrames3_3,4);

%% Test 3
for k = 1 : numFrames3
    mov(k).cdata = vidFrames1_3(:,:,:,k);
    mov(k).colormap = [];
    mov2(k).cdata = vidFrames2_3(:,:,:,k);
    mov2(k).colormap = [];
    mov3(k).cdata = vidFrames3_3(:,:,:,k);
    mov3(k).colormap = [];
end
x3 = zeros(1, numFrames3); y3 = x3;
for jj = 1:numFrames3
   X = rgb2gray(frame2im(mov(jj)));
   X(:, [1:260 400:end]) = 0; % hand calculated X
   X([1:200 389:end], :) = 0; % hand calculated Y
   [mmY, mmX] = find(X > max(X(:)) - 30);
   y3(jj) = mean(mmY);
   x3(jj) = mean(mmX);
end

x31 = zeros(1, numFrames3); y31 = x31;
for jj = 1:numFrames3
   X = rgb2gray(frame2im(mov2(jj)));
   X(:, [1:230 397:end]) = 0; % hand calculated X
   X([1:160 376:end], :) = 0; % hand calculated Y
   [mmY, mmX] = find(X > max(X(:)) - 30);
   y31(jj) = mean(mmY);
   x31(jj) = mean(mmX);
end

x32 = zeros(1, numFrames3); y32 = x32;
for jj = 1:numFrames3
   X = rgb2gray(frame2im(mov3(jj)));
   X(:, [1:148 437:end]) = 0; % hand calculated X
   X([1:211 337:end], :) = 0; % hand calculated Y
   [mmY, mmX] = find(X > max(X(:)) - 30);
   y32(jj) = mean(mmY);
   x32(jj) = mean(mmX);
end

[~, ind] = max(y3(1:30));
x3 = x3(ind:end);
y3 = y3(ind:end);
[~, ind2] = max(y31(1:30));
x31 = x31(ind2:end);
y31 = y31(ind2:end);
[~, ind3] = max(x32(1:30));
x32 = x32(ind3:end);
y32 = y32(ind3:end);

y1 = y3; y11 = y31; y12 = y32;
```

```
x1 = x3; x11 = x31; x12 = x32;

%%
sh = min([length(x1), length(x11), length(x12)]);
x1 = x1(1:end - (length(x1) - sh));  y1 = y1(1:end - (length(y1) - sh));
x11 = x11(1:end - (length(x11) - sh)); y11 = y11(1:end - (length(y11) - sh));
x12 = x12(1:end - (length(x12) - sh)); y12 = y12(1:end - (length(y12) - sh));

%% FFT for smoothing
n = length(y1);
L = length(y1);
k = (2*pi/L)*[0:n/2-1 -n/2:-1];
tau = .2;
k0 = 0;
filter = exp(-tau*(k-k0).^2);
y1 = abs(ifft(filter.*fft(y1)));
y11 = abs(ifft(filter.*fft(y11)));
y12 = abs(ifft(filter.*fft(y12)));
x1 = abs(ifft(filter.*fft(x1)));
x11 = abs(ifft(filter.*fft(x11)));
x12 = abs(ifft(filter.*fft(x12)));

%%
figure(1)
subplot(3,2,1)
plot(y1)
xlabel('Time [Frame]')
ylabel('Position [Y]')
set(gca,'Fontsize',8)
subplot(3,2,2)
plot(x1)
ylabel('Position [X]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
subplot(3,2,3)
plot(y11)
ylabel('Position [Y]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
subplot(3,2,4)
plot( x11)
ylabel('Position [X]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
subplot(3,2,5)
plot(y12)
ylabel('Position [Y]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
subplot(3,2,6)
plot(x12)
ylabel('Position [X]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
```

```
%% SVD
XMAT = [x1; y1; x11; y11; x12; y12];
[m,n]=size(XMAT);   %  compute data size
mn=mean(XMAT,2); %  compute mean for each row
XMAT=XMAT-repmat(mn,1,n);
X = XMAT;

[U,S,V]=svd(X/sqrt(n-1));  % perform the SVD
lambda=diag(S).^2;  % produce diagonal variances
Y = U'*X;% produce the principal components projection
%%
figure(2)
plot(lambda./sum(lambda), 'go', 'MarkerSize', 8);
hold on;
plot(lambda./sum(lambda), 'g--', 'Linewidth',2);
hold off;
set(gca, 'Fontsize', 8, 'Xtick',0:6)
title('Energy by Rank-N Approximation')
xlabel('Rank-N Approximation')
ylabel('Energy')

figure(3)
plot(Y(1,:)); hold on;
plot(Y(2,:));
plot(Y(3,:));
plot(Y(4,:));
plot(Y(5,:));
plot(Y(6,:)); hold off;
set(gca, 'Fontsize', 8)
title('Principal Components [PC]')
legend('PC 1', 'PC 2', 'PC 3', 'PC 4', 'PC 5', ...
    'PC 6', 'Location', 'best')
xlabel('Time [Frame]')
ylabel('Position')

%%%%%%%%%% PART 4 %%%%%%%%%%%%%
close all; clear; clc;
load('cam1_4.mat')
load('cam2_4.mat')
load('cam3_4.mat')

numFrames4 = size(vidFrames1_4,4);

%% Test 4
for k = 1 : numFrames4
    mov(k).cdata = vidFrames1_4(:,:,:,k);
    mov(k).colormap = [];
    mov2(k).cdata = vidFrames2_4(:,:,:,k);
    mov2(k).colormap = [];
    mov3(k).cdata = vidFrames3_4(:,:,:,k);
```

```
    mov3(k).colormap = [];
end
x4 = zeros(1, numFrames4); y4 = x4;
for jj = 1:numFrames4
   X = rgb2gray(frame2im(mov(jj)));
   X(:, [1:320 450:end]) = 0; % hand calculated X
   X([1:230 385:end], :) = 0; % hand calculated Y
   [mmY, mmX] = find(X > max(X(:)) - 20);
   y4(jj) = mean(mmY);
   x4(jj) = mean(mmX);
end

x41 = zeros(1, numFrames4); y41 = x41;
for jj = 1:numFrames4
   X = rgb2gray(frame2im(mov2(jj)));
   X(:, [1:197 380:end]) = 0; % hand calculated X
   X([1:75 356:end], :) = 0; % hand calculated Y
   [mmY, mmX] = find(X > max(X(:)) - 20);
   y41(jj) = mean(mmY);
   x41(jj) = mean(mmX);
end

x42 = zeros(1, numFrames4); y42 = x42;
for jj = 1:numFrames4
   X = rgb2gray(frame2im(mov3(jj)));
   X(:, [1:271 495:end]) = 0; % hand calculated X
   X([1:173 405:end], :) = 0; % hand calculated Y
   [mmY, mmX] = find(X > max(X(:)) - 20);
   y42(jj) = mean(mmY);
   x42(jj) = mean(mmX);
end

[~, ind] = min(y4(1:20));
x4 = x4(ind:end);
y4 = y4(ind:end);
[~, ind2] = min(y41(1:25));
x41 = x41(ind2:end);
y41 = y41(ind2:end);
[~, ind3] = min(x42(1:20));
x42 = x42(ind3:end);
y42 = y42(ind3:end);

y1 = y4; y11 = y41; y12 = y42;
x1 = x4; x11 = x41; x12 = x42;

%%
sh = min([length(x1), length(x11), length(x12)]);
x1 = x1(1:end - (length(x1) - sh));  y1 = y1(1:end - (length(y1) - sh));
x11 = x11(1:end - (length(x11) - sh)); y11 = y11(1:end - (length(y11) - sh));
x12 = x12(1:end - (length(x12) - sh)); y12 = y12(1:end - (length(y12) - sh));

%% FFT for smoothing
n = length(y1);
L = length(y1);
```

```
k = (2*pi/L)*[0:(n-1)/2 -(n-1)/2:-1];
tau = .5;
k0 = 0;
filter = exp(-tau*(k-k0).^2);
y1 = abs(ifft(filter.*fft(y1)));
y11 = abs(ifft(filter.*fft(y11)));
y12 = abs(ifft(filter.*fft(y12)));
x1 = abs(ifft(filter.*fft(x1)));
x11 = abs(ifft(filter.*fft(x11)));
x12 = abs(ifft(filter.*fft(x12)));

%%
figure(1)
subplot(3,2,1)
plot(y1)
xlabel('Time [Frame]')
ylabel('Position [Y]')
set(gca,'Fontsize',8)
subplot(3,2,2)
plot(x1)
ylabel('Position [X]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
subplot(3,2,3)
plot(y11)
ylabel('Position [Y]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
subplot(3,2,4)
plot( x11)
ylabel('Position [X]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
subplot(3,2,5)
plot(y12)
ylabel('Position [Y]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
subplot(3,2,6)
plot(x12)
ylabel('Position [X]')
xlabel('Time [Frame]')
set(gca,'Fontsize',8)
%% SVD
XMAT = [x1; y1; x11; y11; x12; y12];
[m,n]=size(XMAT);   %  compute data size
mn=mean(XMAT,2); %  compute mean for each row
XMAT=XMAT-repmat(mn,1,n);
X = XMAT;

[U,S,V]=svd(X/sqrt(n-1));  % perform the SVD
lambda=diag(S).^2;  % produce diagonal variances
Y = U'*X;% produce the principal components projection
%%
```

```
figure(2)
plot(lambda./sum(lambda), 'go', 'MarkerSize', 8);
hold on;
plot(lambda./sum(lambda), 'g--', 'Linewidth',2);
hold off;
set(gca, 'Fontsize', 8, 'Xtick',0:6)
title('Energy by Rank-N Approximation')
xlabel('Rank-N Approximation')
ylabel('Energy')

figure(3)
plot(Y(1,:)); hold on;
plot(Y(2,:));
plot(Y(3,:));
plot(Y(4,:));
plot(Y(5,:));
plot(Y(6,:)); hold off;
set(gca, 'Fontsize', 8)
title('Principal Components [PC]')
legend('PC 1', 'PC 2', 'PC 3', 'PC 4', 'PC 5', ...
    'PC 6', 'Location', 'best')
xlabel('Time [Frame]')
ylabel('Position')
```

C References

MATH 482 Course Notes