HW3_PCA

Yanjie Liu

Abstract:

The goal of this report is to analyze oscillation movement in given video files by principal component analysis. In the four sets of given videos, each set contains three different video which records the same object from different angles in different locations. Some videos have shaking image, horizontal displacement, or rotation. Specifically, we will display the different performance of principal component analysis in different conditions. Then we will explore the energy content of each principal components and compare the original oscillation behavior with the projected data from significant principal components

Introduction and Overview

For the four given cases, the first case is the ideal condition which only has slight displacement of camera. Also, the oscillation direction is straight in z axis. For the second case, the camera has relatively more drastic displacement which makes the data noisy. The third case is more complicated. The camera is moved horizontally, which makes the oscillation object move in all x, y, and z direction in the video. The fourth case is similar with the third one, but the camera is held with rotation. For each case, we will use the lightest point of the oscillation object to track its motion and store the trajectory in x and y axis. Then we will use singular value decomposition to do the principal component analysis. Then we will figure out which principle component is the most important and the difference between the energy of each principal component.

Theoretical Background

Singular Value Decomposition

A singular value decomposition is a factorization of a matrix into a number of constitutive components all of which have a specific meaning in applications. Different from eigen decomposition, singular value decomposition can decompose the matrix with different number of columns and rows. If we define the singular value decomposition of matrix A which is $m \times \mathrm{n}$ is:

$A = U\Sigma W^T$

U is a matrix with $m \times \mathrm{n}$ and Σ is $m \times m$, which is a diagonal matrix. And V is $n \times n$. In this equation, U and V are both unitary matrices, which means:

$U^T U = I$

$V^T V = I$

We can also compute the eigen decomposition with $A^T A$ and AA^T which are $n \times n$ and $m \times m$ matrices. The eigen vectors in $A^T A$ will compose V in the singular value decomposition and the eigen vector in AA^T will compose U in the singular value decomposition. Then we can use the following equation to solve for the singular value matrix Σ .

$$AV = U\Sigma V^T V = U\Sigma => Av_i = \sigma_i u_i = A\frac{v_i}{u_i}$$

We can also compute the singular value decomposition with

$A^T A = (U\Sigma V^T)^T (U\Sigma V^T)$

$= V\Sigma U^T U\Sigma V^T$

$= V\Sigma^2 V^T$

Then we know that the eigen vectors from $A^T A$ compose U matrix. And the eigen value matrix equals to the singular value matrix. Then we will find that the eigen value equals to the square of the singular value.

$\sigma_i = \sqrt{\lambda_i}$

Principal Component Analysis

Principal component analysis is one of the applications of the singular value decomposition. It is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. It is usually used in low dimension like this report. In this report, the dimension is 6 which is the number of measure dimensions. Then the covariance of the single matrix X is:

$$C_X = \frac{1}{n-1} XX^T$$

C_X is the variance of X along its vectors. Specifically, the diagonal of C_X is the variance of corresponding measurement. The off-diagonal elements of C_X are the covariance between two different measure dimensions

Algorithm Implementation and Development

Set-up

Before the start of analysis, we need to load the video files into Matlab and get the number of frames to operate on each frames.

Find the motion trajectory from video

In order to obtain the specific location of the oscillating object, we will use the lightest point of the object to represent it and the motion of the lightest point represents the motion of the oscillating object. However, in some frames, there are other point lighter than the oscillating object. Then we will use a filter to narrow the boundary of the video, which makes sure that the lightest point are from the oscillating object. In order to do so, I use the filter which is very similar with the Shannon filter in previous homework. In order to compare which point is the lightest, we use rgb2gray() to transform the colorful video to black-and-white video. For the black-and-white video, since the degree of light is a number, we will use max() method to find the lightest point and inequality equation to find those points which is relatively lighter than the most of other points in the same frame. Then we will average the location of those light points, since the lightness of the oscillating object changes over time and its motion. In this way, the motion trajectory of the light point is much closer to the motion trajectory of the oscillating object. Also, since the start time of each video is different for the same case, we will set the maximum height of each video as the start to make sure the three camera in the same case record at the same time.

Apply PCA and plot

In order to apply principal component analysis, we need to put the data from each measurement into a single matrix. Since each measurement has different length, we will choose the shortest one as the length of each vector. Then we need to subtract the mean of each corresponding row from this single matrix. Then we will put those vectors into a single matrix as the input of the svd() method. In the svd() method, we will use a economic-size decomposition which is displayed in lecture. Finally, we will plot the energy percentage of each measurement dimension and the position of each component. We will not use semilogy() method since the difference between each principal component is not too huge. For all the four cases, the process is almost the same.

Computational Result

Test 1 ideal case

In ideal case, we can find that there are one principal component which corresponds to about 65% energy in figure2. The other components have relatively much lower energy. In the video, the oscillating object only has one direction to move, which correspond to the result from principal component analysis.

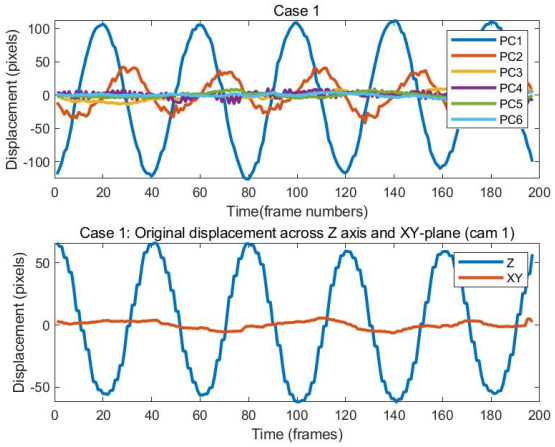
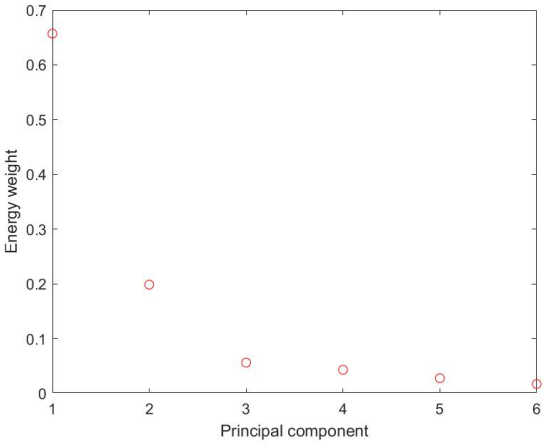
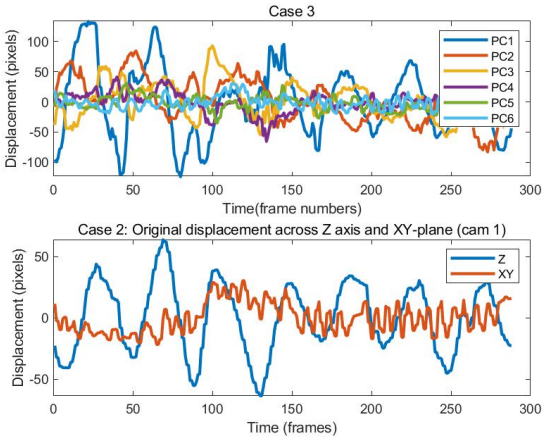


Figure1 (case1):
Plot of principal component
Plot of original displacement



sssFigure2(case1):
Plot of variance of principal components



Figur3(case2):
Plot of principal component
Plot of original displacement

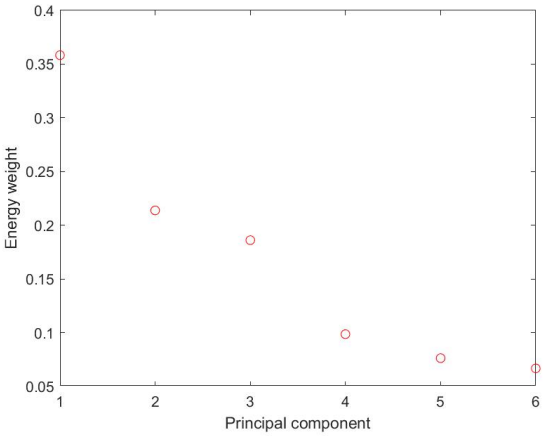
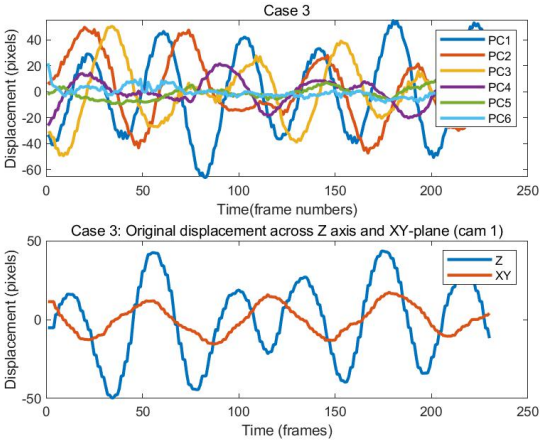


Figure4(case2):
Plot of variance of principal components

Test 2 noisy case

In the noisy case, we can find that there are three principal components which also correspond to about 80% energy in figure4. And components with the second and third large energy weight are much less than the component with largest energy weight. The other components have relatively much lower energy. In the video, since the camera is moved horizontally, the motion of the oscillating object could be seen as two directions, which also correspond to the result from the principal component analysis.



Figur5(case3):
Plot of principal component
Plot of original displacement

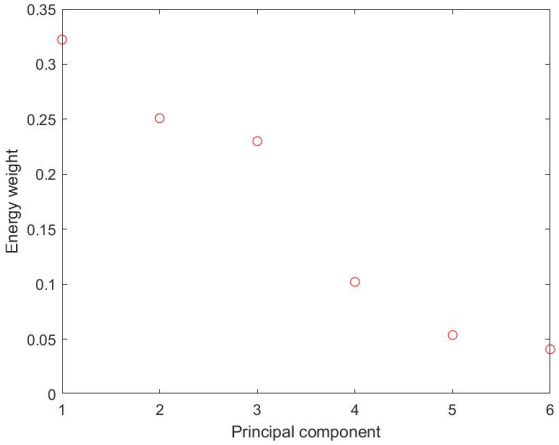
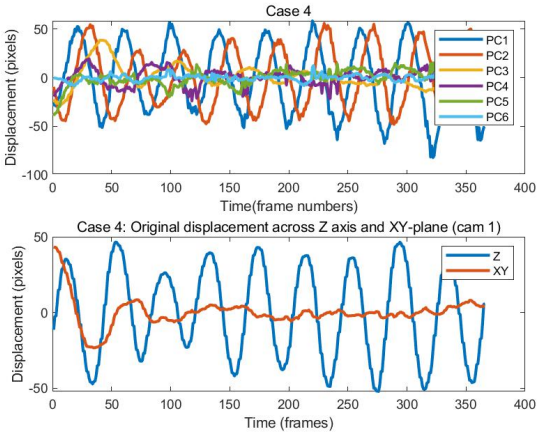


Figure6(case3):
Plot of variance of principal components



Figur7(case4):
Plot of principal component
Plot of original displacement

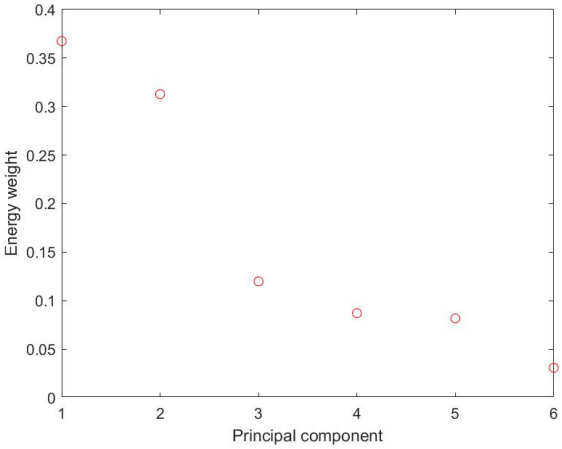


Figure8(case4):
Plot of variance of principal components

Test 3 horizontal displacement

In the case with horizontal displacement , we can find that there are three principal components which also correspond to about 80% energy in figure6. The other components have relatively much lower energy. In the video, since the camera is moved horizontally, the motion of the oscillating object could be seen as two directions, which also correspond to the result from the principal component analysis.

Test 4 horizontal displacement and rotation

In the case with horizontal displacement and rotation, we can find that there are five principal components which also correspond to about 95% energy in figure8. The other components have relatively much lower energy. In the video, since the camera is moved horizontally and vortically with noise, the behavior of the oscillating object greatly correspond with the data that we get.

Summary and Conclusions

In this report , we explore the application of the principal component analysis. Specifically we figure out how to use principal component analysis to figure out the dominant components and energy captured by each component. For the four cases with three cameras, I directly observe the relationship between the dominant component and the motion with only one direction and figure out the practicability of principal component analysis to remove redundant information.

Appendix A

I = rgb2gray(RGB) converts the truecolor image RGB to the grayscale image I. The rgb2gray function converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance.

M = max(A) returns the maximum elements of an array. If A is a matrix, then max(A) is a row vector containing the maximum value of each column.

[row,col] = find(___) returns the row and column subscripts of each nonzero element in array X using any of the input arguments in previous syntaxes.

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.

[M,I] = max(___) also returns the index into the operating dimension that corresponds to the maximum value of A for any of the previous syntaxes.

[U,S,V] = svd(A,'econ') produces an economy-size decomposition of m-by-n matrix A:
m > n — Only the first n columns of U are computed, and S is n-by-n.
m = n — svd(A,'econ') is equivalent to svd(A).
m < n — Only the first m columns of V are computed, and S is m-by-m.

x = diag(A) returns a column vector of the main diagonal elements of A.

description from https://www.mathworks.com/help/matlab/

Appendix B

```
clear all; close all; clc;
load cam1_1;
load cam2_1;
load cam3_1;
load cam1_2;
load cam2_2;
load cam3_2;
load cam1_3;
load cam2_3;
load cam3_3;
load cam1_4;
load cam2_4;
load cam3_4;

numFrames11 = size(vidFrames1_1, 4);
numFrames21 = size(vidFrames2_1, 4);
numFrames31 = size(vidFrames3_1, 4);
numFrames12 = size(vidFrames1_2, 4);
numFrames22 = size(vidFrames2_2, 4);
numFrames32 = size(vidFrames3_2, 4);
numFrames13 = size(vidFrames1_3, 4);
numFrames23 = size(vidFrames2_3, 4);
numFrames33 = size(vidFrames3_3, 4);
numFrames14 = size(vidFrames1_4, 4);
numFrames24 = size(vidFrames2_4, 4);
numFrames34 = size(vidFrames3_4, 4);
```

Test1

```
y11 = [];
x11 = [];

filter = zeros(480,640);
filter(:,300:360) = 1;

for j=1:numFrames11
    gray = rgb2gray(vidFrames1_1(:,:,:,j));
    gray_f = double(gray).*filter;
    [a,~] = max(gray_f(:));
    [X,Y] = find(gray_f > a*11/12);
    x11(j) = [mean(X)];
    y11(j) = [mean(Y)];
end
[a,b] = max(y11(1:30));
y11 = y11(30:end);
x11 = x11(30:end);
```

```
y21 = [];
x21 = [];
```

```
filter = zeros(480,640);
filter(:,250:360) = 1;
for j=1:numFrames21
    gray = rgb2gray(vidFrames2_1(:,:,:,j));
    gray_f = double(gray).*filter;
    [a,~] = max(gray_f(:));
    [X,Y] = find(gray_f > a*11/12);
    x21(j) = [mean(X)];
    y21(j) = [mean(Y)];
end
[a,b] = max(y21(1:40));
y21 = y21(b:end);
x21 = x21(b:end);
```

```
x31 = [];
y31 = [];
filter = zeros(480,640);
filter(200:350,230:490) = 1;
for j=1:numFrames31
    gray = rgb2gray(vidFrames3_1(:,:,:,j));
    gray_f = double(gray).*filter;
    [a,~] = max(gray_f(:));
    [X,Y] = find(gray_f > a*11/12);
    x31(j) = [mean(X)];
    y31(j) = [mean(Y)];
end
[a,b] = max(y31(1:40));
y31 = y31(b:end);
x31 = x31(b:end);
```

Plot

```
x11 = x11(1:min([length(x11),length(x21),length(x31)]));
x21 = x21(1:length(x11));
x31 = x31(1:length(x11));
y11 = y11(1:length(x11));
y21 = y21(1:length(x11));
y31 = y31(1:length(x11));

X = [x11;y11; x21; y21; x31; y31];
[~, n] = size(X);
mn = mean(X, 2);
X = X - repmat(mn,1,n);
[u, s, v] = svd(X, 'econ');
figure()
plot(diag(s)./sum(diag(s)), 'ro')
xlabel('Principal component'); ylabel('Energy weight');
v = v*s;
saveas(gcf,'e1.png');
figure()
subplot(2,1,1)
plot(v(:,1),'Linewidth', 2); hold on;
plot(v(:,2),'Linewidth', 2);
```

```
plot(v(:,3),'Linewidth', 2);
plot(v(:,4),'Linewidth', 2);
plot(v(:,5),'Linewidth', 2);
plot(v(:,6),'Linewidth', 2);
xlabel('Time(frame numbers)')
ylabel('Displacement (pixels)')
title('Case 1')
legend('PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6')
subplot(2,1,2)
plot(1:length(X(1,:)), X(1,:),1:length(X(1,:)), X(2,:),'Linewidth', 2);
ylabel("Displacement (pixels)"); xlabel("Time (frames)");
title("Case 1: Original displacement across Z axis and XY-plane (cam 1)");
legend("Z", "XY")
saveas(gcf,'c1.png');
```

Test2

```
y12 = [];
x12 = [];

filter = zeros(480,640);
filter(170:430,300:450) = 1;

for j=1:numFrames12
    gray = rgb2gray(vidFrames1_2(:,:,:,j));
    gray_f = double(gray).*filter;
    [a,~] = max(gray_f(:));
    [X,Y] = find(gray_f > a*11/12);
    x12(j) = [mean(X)];
    y12(j) = [mean(Y)];
end
[a,b] = max(y12(1:30));
y12 = y12(b:end);
x12 = x12(b:end);
```

```
y22 = [];
x22 = [];

filter = zeros(480,640);
filter(50:475,165:425) = 1;
for j=1:numFrames22
    gray = rgb2gray(vidFrames2_2(:,:,:,j));
    gray_f = double(gray).*filter;
    [a,~] = max(gray_f(:));
    [X,Y] = find(gray_f > a*11/12);
    x22(j) = [mean(X)];
    y22(j) = [mean(Y)];
end
[a,b] = max(y22(1:30));
y22 = y22(b:end);
x22 = x22(b:end);
```

```
x32 = [];
y32 = [];
filter = zeros(480,640);
filter(200:350,230:490) = 1;
for j=1:numFrames32
    gray = rgb2gray(vidFrames3_2(:,:,:,j));
    gray_f = double(gray).*filter;
    [a,~] = max(gray_f(:));
    [X,Y] = find(gray_f > a*11/12);
    x32(j) = [mean(X)];
    y32(j) = [mean(Y)];
end
[a,b] = max(y32(1:40));
y32 = y32(b:end);
x32 = x32(b:end);
```

Plot

```
x12 = x12(1:min([length(x12),length(x22),length(x32)]));
x22 = x22(1:length(x12));
x32 = x32(1:length(x12));
y12 = y12(1:length(x12));
y22 = y22(1:length(x12));
y32 = y32(1:length(x12));

X = [x12; y12; x22; y22; x32; y32];
[~, n] = size(X);
mn = mean(X, 2);
X = X-repmat(mn,1,n);
[u, s, v] = svd(X, 'econ');
figure()
plot(diag(s)./sum(diag(s)), 'ro')
xlabel('Principal component'); ylabel('Energy weight');
v = v*s;
saveas(gcf,'e2.png');
figure()
subplot(2,1,1)
plot(v(:,1),'Linewidth', 2); hold on;
plot(v(:,2),'Linewidth', 2);
plot(v(:,3),'Linewidth', 2);
plot(v(:,4),'Linewidth', 2);
plot(v(:,5),'Linewidth', 2);
plot(v(:,6),'Linewidth', 2);
xlabel('Time(frame numbers)')
ylabel('Displacement (pixels)')
title('Case 3')
legend('PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6')
subplot(2,1,2)
plot(1:length(X(1,:)), X(1,:),1:length(X(1,:)), X(2,:),'Linewidth', 2);
ylabel("Displacement (pixels)"); xlabel("Time (frames)");
title("Case 2: Original displacement across Z axis and XY-plane (cam 1)");
```

```
legend("Z", "XY")
saveas(gcf,'c2.png');
```

Test3

```
y13 = [];
x13 = [];

filter = zeros(480,640);
filter(150:450,200:450) = 1;

for j=1:numFrames13
    gray = rgb2gray(vidFrames1_3(:,:,:,j));
    gray_f = double(gray).*filter;
    [a,~] = max(gray_f(:));
    [X,Y] = find(gray_f > a*11/12);
    x13(j) = [mean(X)];
    y13(j) = [mean(Y)];
end
[~,b] = max(y13(1:30));
y13 = y13(b:end);
x13 = x13(b:end);
```

```
y23 = [];
x23 = [];

filter = zeros(480,640);
filter(100:430,165:425) = 1;
for j=1:numFrames23
    gray = rgb2gray(vidFrames2_3(:,:,:,j));
    gray_f = double(gray).*filter;
    [a,~] = max(gray_f(:));
    [X,Y] = find(gray_f > a*11/12);
    x23(j) = [mean(X)];
    y23(j) = [mean(Y)];
 end
[a,b] = max(y23(1:50));
y23 = y23(b:end);
x23 = x23(b:end);
```

```
x33 = [];
y33 = [];
filter = zeros(480,640);
filter(160:365,235:595) = 1;
for j=1:numFrames33
    gray = rgb2gray(vidFrames3_3(:,:,:,j));
    gray_f = double(gray).*filter;
    [a,~] = max(gray_f(:));
    [X,Y] = find(gray_f > a*11/12);
    x33(j) = [mean(X)];
    y33(j) = [mean(Y)];
end
[a,b] = max(y33(1:30));
```

```
y33 = y33(b:end);
x33 = x33(b:end);
```

Plot

```
x13 = x13(1:min([length(x13),length(x23),length(x33)]));
x23 = x23(1:length(x13));
x33 = x33(1:length(x13));
y13 = y13(1:length(x13));
y23 = y23(1:length(x13));
y33 = y33(1:length(x13));

X = [x13; y13; x23; y23; x33; y33];
[m, n] = size(X);
mn = mean(X, 2);
X = X-repmat(mn,1,n);
[u, s, v] = svd(X, 'econ');
figure()
plot(diag(s)./sum(diag(s)), 'ro')
xlabel('Principal component'); ylabel('Energy weight');
v = v*s;
saveas(gcf,'e3.png');
figure()
subplot(2,1,1)
plot(v(:,1),'Linewidth', 2); hold on;
plot(v(:,2),'Linewidth', 2);
plot(v(:,3),'Linewidth', 2);
plot(v(:,4),'Linewidth', 2);
plot(v(:,5),'Linewidth', 2);
plot(v(:,6),'Linewidth', 2);
xlabel('Time(frame numbers)')
ylabel('Displacement (pixels)')
title('Case 3')
legend('PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6')
subplot(2,1,2)
plot(1:length(X(1,:)), X(1,:),1:length(X(1,:)), X(2,:),'Linewidth', 2);
ylabel("Displacement (pixels)"); xlabel("Time (frames)");
title("Case 3: Original displacement across Z axis and XY-plane (cam 1)");
legend("Z", "XY")
saveas(gcf,'c3.png');
```

Test4

```
y14 = [];
x14 = [];

filter = zeros(480,640);
filter(220:450,270:470) = 1;

for j=1:numFrames14
   gray = rgb2gray(vidFrames1_4(:,:,:,j));
```

```
    gray_f = double(gray).*filter;
    [a,~] = max(gray_f(:));
    [X,Y] = find(gray_f > a*11/12);
    x14(j) = [mean(X)];
    y14(j) = [mean(Y)];
end
[~,b] = max(y14(1:30));
y14 = y14(b:end);
x14 = x14(b:end);
```

```
y24 = [];
x24 = [];

filter = zeros(480,640);
filter(50:400,170:425) = 1;
for j=1:numFrames24
    gray = rgb2gray(vidFrames2_4(:,:,:,j));
    gray_f = double(gray).*filter;
    [a,~] = max(gray_f(:));
    [X,Y] = find(gray_f > a*11/12);
    x24(j) = [mean(X)];
    y24(j) = [mean(Y)];
 end
[a,b] = max(y24(1:30));
y24 = y24(b:end);
x24 = x24(b:end);
```

```
x34 = [];
y34 = [];
filter = zeros(480,640);
filter(100:300,270:510) = 1;
for j=1:numFrames34
    gray = rgb2gray(vidFrames3_4(:,:,:,j));
    gray_f = double(gray).*filter;
    [a,~] = max(gray_f(:));
    [X,Y] = find(gray_f > a*11/12);
    x34(j) = [mean(X)];
    y34(j) = [mean(Y)];
end
[a,b] = max(y34(1:30));
y34 = y34(b:end);
x34 = x34(b:end);
```

Plot

```
x14 = x14(1:min([length(x14),length(x24),length(x34)]));
x24 = x24(1:length(x14));
x34 = x34(1:length(x14));
y14 = y14(1:length(x14));
y24 = y24(1:length(x14));
y34 = y34(1:length(x14));

X = [x14; y14; x24; y24; x34; y34];
```

```
[m, n] = size(X);
mn = mean(X, 2);
X = X-repmat(mn,1,n);
[u, s, v] = svd(X, 'econ');
figure()
plot(diag(s)./sum(diag(s)), 'ro')
xlabel('Principal component'); ylabel('Energy weight');
v = v*s;
saveas(gcf,'e4.png');
figure()
subplot(2,1,1)
plot(v(:,1),'Linewidth', 2); hold on;
plot(v(:,2),'Linewidth', 2);
plot(v(:,3),'Linewidth', 2);
plot(v(:,4),'Linewidth', 2);
plot(v(:,5),'Linewidth', 2);
plot(v(:,6),'Linewidth', 2);
xlabel('Time(frame numbers)')
ylabel('Displacement (pixels)')
title('Case 4')
legend('PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6')
subplot(2,1,2)
plot(1:length(X(1,:)), X(1,:),1:length(X(1,:)), X(2,:),'Linewidth', 2);
ylabel("Displacement (pixels)"); xlabel("Time (frames)");
title("Case 4: Original displacement across Z axis and XY-plane (cam 1)");
legend("Z", "XY")
saveas(gcf,'c4.png');
```