AMATH 482 Homework 3

Rui Zhu

Feb 22, 2021

Abstract

In this project we are given a number of movie files (12) created from three different cameras. 8 of these 12 files contain noise caused by the shakes or rotation of the cameras. We will apply principal component analysis on four systems of an oscillating paint and compare them under 4 different cases.

1 Introduction and Overview

Singular Value Decomposition (SVD) is a way of factoring matrices into three separate components. It is widely used in the field of Data Analysis partly due to its dimensional reductions ability. **Principle Component Analysis** (PCA) is an application of SVD. Generally speaking, we will see what low-rank approximations mean geometrically when we are working with data matrices.

One critical aspect of these videos is that there is a bright spot on the paint. We will use this bright spot to track the movement of the paint. We record the xy coordinates of the bright spot and combine them into a large matrix. We then perform SVD to this matrix to see which principle components are the most dominant, and how much energy they captured.

There are 4 cases in this assignment. The first case is relatively ideal: there is a small displacement of the mass in the z direction and the ensuing oscillations. In this case, the entire motion is in the z direction with simple harmonic motion being observed; Case 2 introduce camera shake; In Case 3 the mass is released off-centre so as to produce motion in the xy plane as well as the z direction; In Case 4 the mass is released off-centre and rotates so as to produce rotation in the xy plane and motion in the z direction.

2 Theoretical Background

2.1 SVD

SVD factors a matrix into a product of three matrices. For example, if we apply SVD to a matrix A We have $A = U\Sigma V^*$ where U and V are unitary matrices that simply lead to rotation and Σ is a diagonal matrix that scales an image as prescribed by the singular values. There are many benefits of performing SVD, a major one is to remove redundancy and describe the signals or data with a maximal variance.

3 Algorithm Implementation and Development

Our algorithm will be very similar for all 4 cases:

3.1 Initial Setup

1. Read the 3 video data for current case.
2. Obtain the number of frames, height, and width of each videos.

3.2 Filtering

1. Define our filters to remove irrelevant part (keep the paint in the video only)
2. Adjust the locations of our filters until we only keep the paint in the screen, for example:

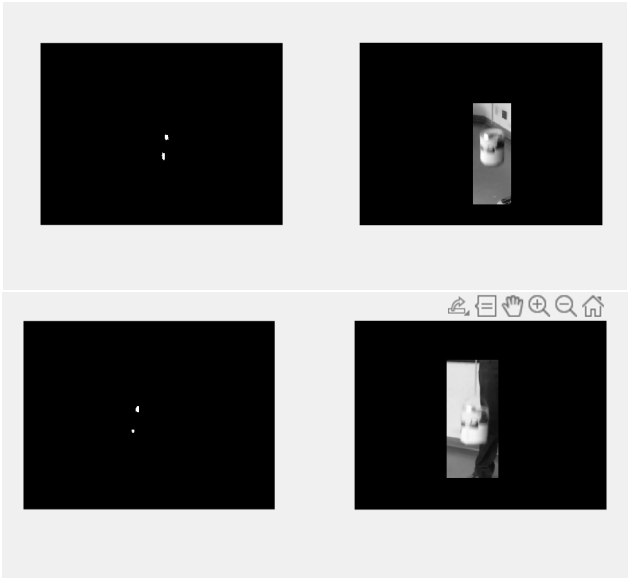

Figure 1: Final filters for camera 1.1 and camera 1.2

3.3 SVD and Plotting

1. Check each frames individually

Algorithm 1:

Data: `currentcam.mat`

for *j in frame* **do**
- Convert current frame to grayscale to check the bright spot using `frame2im()`, `rgb2grey()` ,and `double()`
- Track the bright spot using a threshold matrix where the value is greater than 245
- Use `find()` and `ind2sub()` to locate these bright spots, then save them to our current camera array

2. Repeat same procedure for the other two cameras' data

3. Use the shortest array as 'standard' to trim off the other two arrays so they all have the same number of frames

4. Subtract the mean of each row to center the data

5. Apply SVD to our matrix using `svd()`, notice that we need to divide our matrix by $\sqrt{n-1}$

6. Plot the normalized variances to capture the significant principal components.

7. Plot the projections of our data and repeat for the other 3 sets of data.

4 Computational Results

Notice that in each Test we need to adjust our filter to keep only the paint in the screen.

4.1 Test 1: Ideal Case

As we can see in Figure 2 below, the first principal component captured more than 90% of energy. This meets our expectation: since the paint only oscillates in one single direction, one single principal component should be enough. Additionally, our projection on to the first principal component basis is very similar to what we measured from the actual data, so this system: Case 1 is successfully reproduced and represented using PCA.

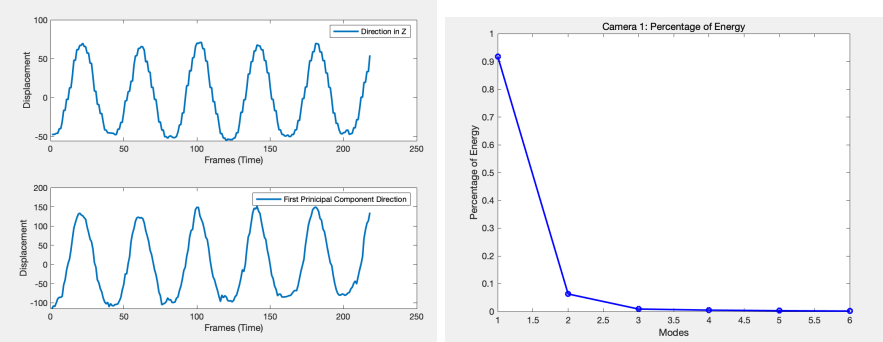

Figure 2: Plot of original displacement VS displacement under principal component dierctions

4.2 Test 2: Noisy Case

As we can see in Figure 3 below, the first principal component captured more than 60% of energy, and the second principal component captured more than 20% of energy. Thus, in this test we need two principal components for this system. Although Test 2 is basically Test 1 plus noise, our PCA did not perform well for this test, the noise consistently shows up in each frame during the paint's oscillation. But we can still observed an oscillatory behavior (generally speaking).

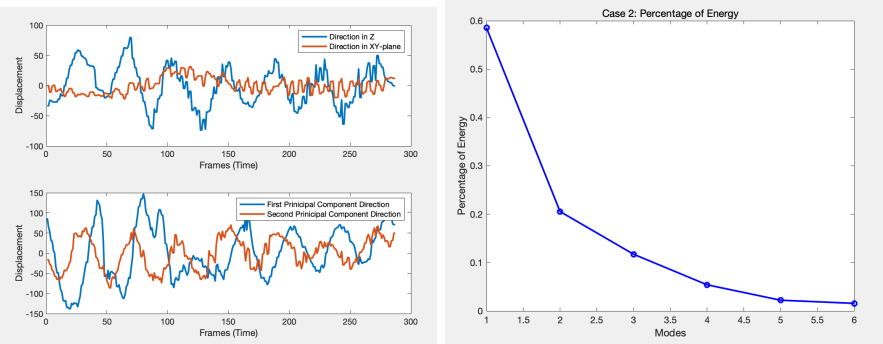

Figure 3: Plot of original displacement VS displacement under principal component dierctions

4.3 Test 3: Horizontal Displacement

As we can see in Figure 4 below, the first principal component captured only more than 50% of energy, the second principal component captured more than 25% of energy, the third principal component captured more than 15% of energy, and the fourth principal component captured more than 10% energy. Thus, in this test we need four principal components for this system. Although the paint is released off-centre, we can still see the curves in the left hand side is quite smooth, thus there is little error caused by noise in this system, we observed a clear oscillatory behavior.

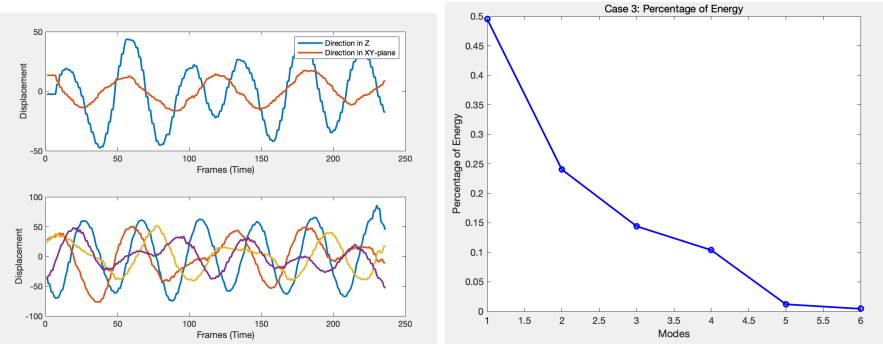

Figure 4: Plot of original displacement VS displacement under principal component dierctions

4.4 Test 4: Horizontal Displacement and Rotation

Surprisingly, as we can see in Figure 5 below, the first principal component captured around 65% of energy, the second principal component captured around 25% of energy, and the third principal component captured more than 15% of energy. Thus, in this test we only need three principal components for this system. Notice the large displacement in XY-plane in the beginning of the video caused by the off-centering. Again there's little error caused by noise in this system, and we observed a clear oscillatory behavior.

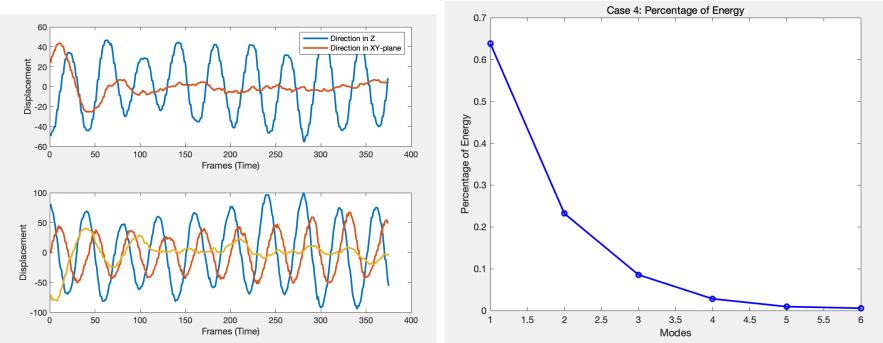

Figure 5: Plot of original displacement VS displacement under principal component dierctions

5 Summary and Conclusions

We implemented PCA to 4 different cases of an oscillating paint to find the number of significant principal components in each system. Surprisingly, PCA performed poorly for Test 3 where the paint is released off-centre so as to produce motion in the xy plane as well as the z direction; Whereas PCA performed better in Test 4 (Test 3 plus Rotation). This situation might be caused by a poorly selected filter. But generally speaking, we successfully reduced the dimension of the original matrix and produced relatively accurate projections.

Appendix A MATLAB Functions

- `[X,Y,Z]=ind2sub([n n n] index)` returns a 1*3 array based on the input dimensions and a 1D index.
- `RGB = frame2im(F)` returns the truecolor (RGB) image from the single movie frame F.
- `I = rgb2gray(RGB)` converts the truecolor image RGB to the grayscale image I by eliminating the hue and saturation information while retaining the luminance.
- `k = find(X)` returns a vector containing the linear indices of each nonzero element in array X.
- `M = mean(A,dim)` returns the mean along dimension dim. For example, if A is a matrix, then mean(A,2) is a column vector containing the mean of each row.
- `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.
- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix A, such that $A = U * \Sigma * V'$.

Appendix B MATLAB Code

```
clear all; close all; clc
%% Load & Watch movie (Sample code from Slack)
% -- Load Movie
% load cam1_1.mat;
% [height width rgb num_frames] = size(vidFrames1_1);
%
% frame = getframe(gcf);
%
% for j=1:num_frames3_1
%     X=vidFrames3_1(:,:,:,j);
%     imshow(X);
%     drawnow
% end

%% Test 1: Ideal Case
```

```
load cam1_1.mat
load cam2_1.mat
load cam3_1.mat

[height1_1 width1_1 rgb1_1 num_frames1_1] = size(vidFrames1_1);
[height2_1 width2_1 rgb2_1 num_frames2_1] = size(vidFrames2_1);
[height3_1 width3_1 rgb3_1 num_frames3_1] = size(vidFrames3_1);



for j=1:num_frames1_1
    movie1(j).cdata=vidFrames1_1(:,:,:,j);
    movie1(j).colormap=[];
end

for j=1:num_frames2_1
    movie2(j).cdata=vidFrames2_1(:,:,:,j);
    movie2(j).colormap=[];
end

for j=1:num_frames3_1
    movie3(j).cdata=vidFrames3_1(:,:,:,j);
    movie3(j).colormap=[];

end

%% filter out the paint for camera 1

filter_width = 50;
filter1 = zeros(height1_1, width1_1);
filter1(300-2.8*filter_width : 1 : 300+2.5*filter_width, ...
        350-1*filter_width : 1 : 350+1*filter_width) = 1;

video1 = [];
```

```
for i = 1:num_frames1_1
    a1 = double(rgb2gray(frame2im(movie1(i))));
    a1_f = a1.* filter1;
    thr = a1_f > 250;
    ind = find(thr);
    [Y,X] = ind2sub(size(thr), ind);

    video1 = [video1; mean(X), mean(Y)];

%     subplot(1,2,1)
%     imshow(uint8(thr*250)); drawnow
%     subplot(1,2,2)
%     imshow(uint8(a1_f)); drawnow
end
% close all;

video2 =[];
filter2 = zeros(height2_1, width2_1);
filter2(250-3*filter_width : 1 : 250+3*filter_width, ...
        300-1.3*filter_width : 1 : 300+1.3*filter_width) = 1;

for i = 1:num_frames2_1
    a2 = double(rgb2gray(frame2im(movie2(i))));
    a2_f = a2.* filter2;
    thr = a2_f > 250;
    ind = find(thr);
    [Y,X] = ind2sub(size(thr), ind);

    video2 = [video2; mean(X), mean(Y)];

%     subplot(1,2,1)
%     imshow(uint8(thr*255)); drawnow
```

```
%     subplot(1,2,2)
%     imshow(uint8(a2_f)); drawnow
end

video3 =[];
filter3 = zeros(height3_1, width3_1);
filter3(250-1*filter_width : 1 : 250+2*filter_width, ...
        360-2.5*filter_width : 1 : 360+2.5*filter_width) = 1;

for i = 1:num_frames3_1
    a3 = double(rgb2gray(frame2im(movie3(i))));
    a3_f = a3.* filter3;
    thr = a3_f > 246; %adjust threshold value here to avoid NaN values
    ind = find(thr);
    [Y,X] = ind2sub(size(thr), ind);

    video3 = [video3; mean(X), mean(Y)];

%     subplot(1,2,1)
%     imshow(uint8(thr*255)); drawnow
%     subplot(1,2,2)
%     imshow(uint8(a3_f)); drawnow
end

[M, I] = min(video1(1:20,2));
video1 = video1(I:end, :);

[M, I] = min(video2(1:20,2));
video2 = video2(I:end, :);
video2 = video2(1:length(video1),:);

[M, I] = min(video3(1:20,2));
video3 = video3(I:end, :);
```

```
video3 = video3(1:length(video1),:);

%% SVD for camera 1
cam1 = [video1' ; video2' ; video3'];
[m,n] = size(cam1);



cam1_mean = mean(cam1,2); % compute mean by rows
cam1 = cam1 - repmat(cam1_mean,1,n);

[u,s,v] = svd(cam1' / sqrt(n-1)); %perform SVD
sig = diag(s);
lam = diag(s) .^ 2; % diagonal variances
Y = cam1' * v;

%%
figure()
plot(1:1:m, lam/sum(lam), 'bo-', 'Linewidth', 2);
title("Camera 1: Percentage of Energy");
xlabel("Modes");
ylabel("Percentage of Energy");

figure()
title("Case 1: Displacement across Z axis");

subplot(2,1,1)
plot(1:218, cam1(2,:), 'Linewidth', 2);
xlabel("Frames (Time)");
ylabel("Displacement");
legend("Direction in Z", "First Prinicipal Component Direction");

subplot(2,1,2)
plot(1:218, Y(:,1), 'Linewidth', 2);
```

```
xlabel("Frames (Time)");
ylabel("Displacement");
legend("First Prinicipal Component Direction");

%% Test 2

load cam1_2.mat
load cam2_2.mat
load cam3_2.mat

[height1_2 width1_2 rgb1_2 num_frames1_2] = size(vidFrames1_2);
[height2_2 width2_2 rgb2_2 num_frames2_2] = size(vidFrames2_2);
[height3_2 width3_2 rgb3_2 num_frames3_2] = size(vidFrames3_2);


for j=1:num_frames1_2
    movie1(j).cdata=vidFrames1_2(:,:,:,j);
    movie1(j).colormap=[];
end

for j=1:num_frames2_2
    movie2(j).cdata=vidFrames2_2(:,:,:,j);
    movie2(j).colormap=[];
end

for j=1:num_frames3_2
    movie3(j).cdata=vidFrames3_2(:,:,:,j);
    movie3(j).colormap=[];
end


%% filter out the paint for camera 1
```

```
filter_width = 50;
filter1 = zeros(height1_2, width1_2);
filter1(300-2.8*filter_width : 1 : 300+2.5*filter_width, ...
        350-1*filter_width : 1 : 350+1*filter_width) = 1;

video1 = [];

for i = 1:num_frames1_2
    a1 = double(rgb2gray(frame2im(movie1(i))));
    a1_f = a1.* filter1;
    thr = a1_f > 250;
    ind = find(thr);
    [Y,X] = ind2sub(size(thr), ind);

    video1 = [video1; mean(X), mean(Y)];

%     subplot(1,2,1)
%     imshow(uint8(thr*250)); drawnow
%     subplot(1,2,2)
%     imshow(uint8(a1_f)); drawnow
end
% close all;

video2 =[];
filter2 = zeros(height2_2, width2_2);
filter2(250-4*filter_width : 1 : 250+4.5*filter_width, ...
        290-2.5*filter_width : 1 : 290+2.7*filter_width) = 1;

for i = 1:num_frames2_2
    a2 = double(rgb2gray(frame2im(movie2(i))));
    a2_f = a2.* filter2;
    thr = a2_f > 245;
    ind = find(thr);
```

```
    [Y,X] = ind2sub(size(thr), ind);

    video2 = [video2; mean(X), mean(Y)];

%     subplot(1,2,1)
%     imshow(uint8(thr*255)); drawnow
%     subplot(1,2,2)
%     imshow(uint8(a2_f)); drawnow
end

video3 =[];
filter3 = zeros(height3_2, width3_2);
filter3(250-1*filter_width : 1 : 250+2*filter_width, ...
        360-2.5*filter_width : 1 : 360+2.5*filter_width) = 1;

for i = 1:num_frames3_2
    a3 = double(rgb2gray(frame2im(movie3(i))));
    a3_f = a3.* filter3;
    thr = a3_f > 245; %adjust threshold value here to avoid NaN values
    ind = find(thr);
    [Y,X] = ind2sub(size(thr), ind);

    video3 = [video3; mean(X), mean(Y)];

%     subplot(1,2,1)
%     imshow(uint8(thr*255)); drawnow
%     subplot(1,2,2)
%     imshow(uint8(a3_f)); drawnow
end

[M, I] = min(video1(1:20,2));
video1 = video1(I:end, :);
```

```
[M, I] = min(video2(1:20,2));
video2 = video2(I:end, :);
video2 = video2(1:length(video1),:);

[M, I] = min(video3(1:20,2));
video3 = video3(I:end, :);
video3 = video3(1:length(video1),:);

%% SVD for camera 2
cam2 = [video1' ; video2' ; video3'];
[m,n] = size(cam2);



cam1_mean = mean(cam2,2); % compute mean by rows
cam2 = cam2 - repmat(cam1_mean,1,n);

[u,s,v] = svd(cam2' / sqrt(n-1)); %perform SVD
sig = diag(s);
lam = diag(s) .^ 2; % diagonal variances
Y = cam2' * v;

%%
figure()
plot(1:1:m, lam/sum(lam), 'bo-', 'Linewidth', 2);
title("Case 2: Percentage of Energy");
xlabel("Modes");
ylabel("Percentage of Energy");

figure()
title("Case 2: Displacement across Z axis");

subplot(2,1,1)
plot(1:287, cam2(2,:),1:287, cam2(1,:),'Linewidth', 2);
```

```
xlabel("Frames (Time)");
ylabel("Displacement");
legend("Direction in Z", "Direction in XY-plane");

subplot(2,1,2)
plot(1:287, Y(:,1),1:287, Y(:,2), 'Linewidth', 2);
xlabel("Frames (Time)");
ylabel("Displacement");
legend("First Prinicipal Component Direction", "Second Prinicipal Component Direction");

%% Test 3

load cam1_3.mat
load cam2_3.mat
load cam3_3.mat

[height1_3 width1_3 rgb1_3 num_frames1_3] = size(vidFrames1_3);
[height2_3 width2_3 rgb2_3 num_frames2_3] = size(vidFrames2_3);
[height3_3 width3_3 rgb3_3 num_frames3_3] = size(vidFrames3_3);


for j=1:num_frames1_3
    movie1(j).cdata=vidFrames1_3(:,:,:,j);
    movie1(j).colormap=[];
end

for j=1:num_frames2_3
    movie2(j).cdata=vidFrames2_3(:,:,:,j);
    movie2(j).colormap=[];
end

for j=1:num_frames3_3
    movie3(j).cdata=vidFrames3_3(:,:,:,j);
```

```
    movie3(j).colormap=[];
end


%% filter out the paint for camera 1

filter_width = 50;
filter1 = zeros(height1_3, width1_3);
filter1(300-1.4*filter_width : 1 : 300+3.1*filter_width, ...
        350-1.4*filter_width : 1 : 350+2*filter_width) = 1;

video1 = [];

for i = 1:num_frames1_3
    a1 = double(rgb2gray(frame2im(movie1(i))));
    a1_f = a1.* filter1;
    thr = a1_f > 250;
    ind = find(thr);
    [Y,X] = ind2sub(size(thr), ind);

    video1 = [video1; mean(X), mean(Y)];

%     subplot(1,2,1)
%     imshow(uint8(thr*250)); drawnow
%     subplot(1,2,2)
%     imshow(uint8(a1_f)); drawnow
end
% close all;

video2 =[];
filter2 = zeros(height2_3, width2_3);
filter2(250-3*filter_width : 1 : 250+3.6*filter_width, ...
        290-2.5*filter_width : 1 : 290+2.8*filter_width) = 1;
```

```
for i = 1:num_frames2_3
    a2 = double(rgb2gray(frame2im(movie2(i))));
    a2_f = a2.* filter2;
    thr = a2_f > 245;
    ind = find(thr);
    [Y,X] = ind2sub(size(thr), ind);

    video2 = [video2; mean(X), mean(Y)];

%     subplot(1,2,1)
%     imshow(uint8(thr*255)); drawnow
%     subplot(1,2,2)
%     imshow(uint8(a2_f)); drawnow
end

video3 =[];
filter3 = zeros(height3_3, width3_3);
filter3(250-1*filter_width : 1 : 250+2*filter_width, ...
        360-2.5*filter_width : 1 : 360+2.5*filter_width) = 1;

for i = 1:num_frames3_3
    a3 = double(rgb2gray(frame2im(movie3(i))));
    a3_f = a3.* filter3;
    thr = a3_f > 245; %adjust threshold value here to avoid NaN values
    ind = find(thr);
    [Y,X] = ind2sub(size(thr), ind);

    video3 = [video3; mean(X), mean(Y)];

%     subplot(1,2,1)
%     imshow(uint8(thr*255)); drawnow
%     subplot(1,2,2)
```

```
%     imshow(uint8(a3_f)); drawnow
end

[M, I] = min(video1(1:20,2));
video1 = video1(I:end, :);

[M, I] = min(video2(1:20,2));
video2 = video2(I:end, :);

[M, I] = min(video3(1:20,2));
video3 = video3(I:end, :);
video1 = video1(1:length(video3),:);
video2 = video2(1:length(video3),:);


%% SVD for camera 2
cam3 = [video1' ; video2' ; video3'];
[m,n] = size(cam3);


cam1_mean = mean(cam3,2); % compute mean by rows
cam3 = cam3 - repmat(cam1_mean,1,n);

[u,s,v] = svd(cam3' / sqrt(n-1)); %perform SVD
sig = diag(s);
lam = diag(s) .^ 2; % diagonal variances
Y = cam3' * v;

%%
figure()
plot(1:1:m, lam/sum(lam), 'bo-', 'Linewidth', 2);
title("Case 4: Percentage of Energy");
xlabel("Modes");
```

```
ylabel("Percentage of Energy");


figure()
title("Case 4: Displacement across Z axis");

subplot(2,1,1)
plot(1:236, cam3(2,:),1:236, cam3(1,:),'Linewidth', 2);
xlabel("Frames (Time)");
ylabel("Displacement");
legend("Direction in Z", "Direction in XY-plane");

subplot(2,1,2)
plot(1:236, Y(:,1),1:236, Y(:,2), 1:236, Y(:,3),1:236, Y(:,4),'Linewidth', 2);
xlabel("Frames (Time)");
ylabel("Displacement");

%% Test 4

load cam1_4.mat
load cam2_4.mat
load cam3_4.mat

[height1_4 width1_4 rgb1_4 num_frames1_4] = size(vidFrames1_4);
[height2_4 width2_4 rgb2_4 num_frames2_4] = size(vidFrames2_4);
[height3_4 width3_4 rgb3_4 num_frames3_4] = size(vidFrames3_4);


for j=1:num_frames1_4
    movie1(j).cdata=vidFrames1_4(:,:,:,j);
    movie1(j).colormap=[];
end

for j=1:num_frames2_4
```

```
    movie2(j).cdata=vidFrames2_4(:,:,:,j);
    movie2(j).colormap=[];
end

for j=1:num_frames3_4
    movie3(j).cdata=vidFrames3_4(:,:,:,j);
    movie3(j).colormap=[];
end


%% filter out the paint for camera 1

filter_width = 50;
filter1 = zeros(height1_4, width1_4);
filter1(300-1.4*filter_width : 1 : 300+3.1*filter_width, ...
        350-1.4*filter_width : 1 : 350+2.5*filter_width) = 1;

video1 = [];

for i = 1:num_frames1_4
    a1 = double(rgb2gray(frame2im(movie1(i))));
    a1_f = a1.* filter1;
    thr = a1_f > 245;
    ind = find(thr);
    [Y,X] = ind2sub(size(thr), ind);

    video1 = [video1; mean(X), mean(Y)];

%     subplot(1,2,1)
%     imshow(uint8(thr*250)); drawnow
%     subplot(1,2,2)
%     imshow(uint8(a1_f)); drawnow
end
```

```
% close all;

video2 =[];
filter2 = zeros(height2_4, width2_4);
filter2(250-4*filter_width : 1 : 250+3.6*filter_width, ...
        290-2.5*filter_width : 1 : 290+2.8*filter_width) = 1;

for i = 1:num_frames2_4
    a2 = double(rgb2gray(frame2im(movie2(i))));
    a2_f = a2.* filter2;
    thr = a2_f > 245;
    ind = find(thr);
    [Y,X] = ind2sub(size(thr), ind);

    video2 = [video2; mean(X), mean(Y)];

%     subplot(1,2,1)
%     imshow(uint8(thr*255)); drawnow
%     subplot(1,2,2)
%     imshow(uint8(a2_f)); drawnow
end

video3 =[];
filter3 = zeros(height3_4, width3_4);
filter3(250-3*filter_width : 1 : 250+1*filter_width, ...
        360-1.7*filter_width : 1 : 360+3*filter_width) = 1;

for i = 1:num_frames3_4
    a3 = double(rgb2gray(frame2im(movie3(i))));
    a3_f = a3.* filter3;
    thr = a3_f > 234; %adjust threshold value here to avoid NaN values
    ind = find(thr);
    [Y,X] = ind2sub(size(thr), ind);
```

```
    video3 = [video3; mean(X), mean(Y)];

%     subplot(1,2,1)
%     imshow(uint8(thr*255)); drawnow
%     subplot(1,2,2)
%     imshow(uint8(a3_f)); drawnow
end

[M, I] = min(video1(1:20,2));
video1 = video1(I:end, :);

[M, I] = min(video2(1:20,2));
video2 = video2(I:end, :);

[M, I] = min(video3(1:20,2));
video3 = video3(I:end, :);
video2 = video2(1:length(video3),:);
video1 = video1(1:length(video3),:);


%% SVD for camera 2
cam4 = [video1' ; video2' ; video3'];
[m,n] = size(cam4);


cam4_mean = mean(cam4,2); % compute mean by rows
cam4 = cam4 - repmat(cam4_mean,1,n);

[u,s,v] = svd(cam4' / sqrt(n-1)); %perform SVD
sig = diag(s);
lam = diag(s) .^ 2; % diagonal variances
Y = cam4' * v;
```

```
%%
figure()
plot(1:1:m, lam/sum(lam), 'bo-', 'Linewidth', 2);
title("Case 4: Percentage of Energy");
xlabel("Modes");
ylabel("Percentage of Energy");

figure()
title("Case 24 Displacement across Z axis");

subplot(2,1,1)
plot(1:375, cam4(2,:),1:375, cam4(1,:),'Linewidth', 2);
xlabel("Frames (Time)");
ylabel("Displacement");
legend("Direction in Z", "Direction in XY-plane");

subplot(2,1,2)
plot(1:375, Y(:,1),1:375, Y(:,2), 1:375, Y(:,3),'Linewidth', 2);
xlabel("Frames (Time)");
ylabel("Displacement");
```