AMATH 482 Homework 3

Seoyoung Cho

February 24, 2021

Abstract

Principal Component Analysis (PCA) is a powerful method that is widely used in data analysis to reduce the dimensionality of large data sets. In this assignment, we will explore one use case of PCA to obtain the information of a can bouncing up and down only using the information from three recordings at different angles. With the three recordings, we are given 6 dimensions of x-y data that captures the motion of the can and will perform PCA to model the z-direction motion re-expressed with the most meaningful information.

1 Introduction and Overview

In this assignment, we consider a general situation where three cameras are recording the motion of a can bouncing up and down at a different angle. The task is subdivided into 4 cases: ideal, noisy, horizontal displacement, and horizontal displacement with rotation.

1. Ideal case illustrates the straight and clean up and down motion of the mass.
2. Noisy case illustrates the straight up and down motion including slight horizontal vibration of cameras.
3. Horizontal displacement case illustrates the motion of the mass oscillating both in vertical and horizontal directions.
4. Horizontal displacement with rotation case illustrates the motion of the mass rotating while moving in all x,y, and z-direction.

We are given three recordings of each case, a total of 12 recordings. The motivation of this assignment is to model the actual motion of the mass with only the x-y position obtained from the recordings. Considering the x, y-direction of the motion in all three recordings, the final data will be in 6 by the time dimension. With the given data, we perform Principal Component Analysis to find out how much the data dimension could reduce to, as well as capturing the actual motion of the mass. We then compare the results from all 4 cases and discuss the effect of noise and the usefulness of Principal Component Analysis (PCA).

2 Theoretical Background

This section explains the mathematical concepts required to proceed with the task.

2.1 Singular Value Decomposition (SVD)

A fundamental technique that PCA outgrows from is Singular Value Decomposition (SVD). SVD is simply a technique to decompose any matrix into three matrices. The three matrices essentially re-express the original matrix in terms of new basis that capture its important properties. The SVD matrix is as below:

$$A = U\Sigma V^* \tag{1}$$

Each component takes the form of the following:

$$\begin{bmatrix} \\ A \\ \\ \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & \cdots & u_n \\ & & & \end{bmatrix} \cdot \left[\begin{array}{ccc} \sigma_1 & & \\ & \ddots & \\ & & \sigma_m \\ \hline & 0 & \end{array}\right] \cdot \begin{bmatrix} \\ v_1^* \\ v_2^* \\ \vdots \\ v_n^* \\ \\ \end{bmatrix} \tag{2}$$

If the elements of the original matrix A are complex numbers, the V^* is the conjugate transpose and only the transpose otherwise. For this assignment, we only deal with real numbers, so V^* and V^T are interchangeable. With $A \in \mathbb{R}^{m \times n}$:

- $V \in \mathbb{R}^{n \times n}$, unitary/orthogonal matrix with columns of right singular vectors that contains information about the row space of A.
- $\Sigma \in \mathbb{R}^{m \times n}$, diagonal matrix of σ_i being the singular values ordered from the largest to the smallest: $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_m$. They measure the importance of each columns of U and V.
- $U \in \mathbb{R}^{m \times m}$, unitary/orthogonal matrix with columns of left singular vectors that contains information about the column space of A.

One thing to note is that when A has m columns, it only has at most rank m. Hence, we can select the first m columns, σ_is, and rows from U, Σ, V respectively. This reduction is often called economy SVD and will be performed in MATLAB as `svd(A, 'econ')` to increase the computational efficiency. Apart from the decomposition, another representation of SVD can be obtained by multiplying individual components. If we expand the matrix multiplication, we get: $\sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_m u_m v_m^T$, with each product being the rank 1 matrix that sum up to increasingly improve the approximation of A. To clarify, $\sigma_1 u_1 v_1^T$ makes the best rank 1 approximation of A, $\sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T$ makes the best rank 2 approximation, and so on. What this means is that we can truncate the SVD representation to obtain the low-rank approximation, containing the most important information, of the original matrix. This tactic is commonly applied to large data sets for efficiency, and such application is called Principal Component Analysis.

2.2 Principal Component Analysis (PCA)

As briefly mentioned above, PCA is a method to reduce the dimensionality of a large data set. It could also be thought of as a statistical interpretation of SVD. Considering the basic setup for our assignment as illustrated in figure 5 in Appendix C, we know that a mass bouncing up and down is modeled by the Hooke's Law with the governing equation of 2nd order differential equation:

$$\frac{d^2 f(t)}{dt^2} = -\omega^2 f(t) \tag{3}$$

where $f(t)$ is the function of the z-direction motion of the mass in time. The solution for the above differential equation is:

$$f(t) = c_1 cos(\omega_0 t) + c_2 sin(\omega_0 t) = A cos(\omega t + \omega_0) \tag{4}$$

where A is the amplitude, ω_0 is the initial angular velocity, and some constants c_1, c_2 . Although this equation gives the most accurate z-direction motion of the mass, it doesn't take account of any real-world circumstances such as noise and vibrations. To take account of those, we use PCA to model the motion by extracting governing equation from three camera recordings. Given three recordings at different angles, our data matrix X takes the form of the following:

$$X = \begin{bmatrix} -- & x_1 & -- \\ -- & y_1 & -- \\ -- & x_2 & -- \\ -- & y_2 & -- \\ -- & x_3 & -- \\ -- & y_3 & -- \end{bmatrix} \tag{5}$$

Each row will be the x and y position of the mass from three cameras by time. Hence, x_1 and y_1 tells the position of the mass recorded by camera 1. One thing to note is that although PCA is just another interpretation of SVD with the same data matrix setup, there is an additional step to be taken. For each row of the data matrix, we subtract the mean of each row to zero-center the average positions. Hence, the data matrix X for our assignment will actually be X in equation 5 being zero-centered. Further, there is an issue of redundancy with the data matrix. We know that using Hooke's Law, the motion of the mass can be modeled by 2 sinusoidal equations which only require 2-dimension data. The 6-dimension of X is definitely oversampled. This is where statistical intuition is applied. To detect the redundancy in data, we introduce a covariance matrix of X. The covariance is simply a measure of statistical dependence/independence between two variables. Cx, the covariance matrix is as follows:

$$C_x = \frac{1}{n-1} XX^T \tag{6}$$

$C_x \in \mathbb{R}^{m \times m}$ is a symmetric matrix that captures the correlation between every pair of variables. The diagonal entries are the variances, and the off-diagonal entries are the covariances between the variables. Higher covariance means there are more redundancy and larger variance means the variable fluctuates more, represented as the dynamics of interest. To remove the redundancy and find the dynamics of interest, we diagonalize the covariance matrix. Diagnoalizing C_x will order the variances from the largest to the smallest and set its off-diagonals as 0. This process exactly resembles performing SVD on X considering that σ_i^2 is the variance. Below describes the relationship between SVD and C_x:

$$\begin{aligned} C_x = \frac{XX^T}{n-1} &= \frac{1}{n-1}(U\Sigma V^*)(V\Sigma U^*)[eq1] \\ &= \frac{1}{n-1} U\Sigma^2 U^* \end{aligned}$$

Using SVD matches the columns of U and V to the order of σ_i, or basically the variance, from largest to the smallest. The columns of U represent the principal component modes. They are the new orthonormal basis, linear combinations, of the raw measurements containing the most important information that we can re-express our data as. We can think of the product $U\Sigma$ as representations of the coefficients c_1, c_2 in the governing equation [eq 4]. Finally, the columns of V represent the time dynamics of the corresponding principal component modes that capture the oscillatory motion of the mass. Further, recall that the singular values give the best rank-N approximation of the original matrix. We can also compute the energy of each rank-N approximation of full r rank matrix with the equation below:

$$energy_N = \frac{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_N^2}{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_r^2} \tag{7}$$

Looking at singular values or the energies, we can find the necessary dimension or the principal component modes for the data can be reduced to. Of course, reducing the data dimension would decrease the accuracy of the model. The general idea of PCA is to trade the data dimension with accuracy for more efficient data analysis performance.

3 Algorithm Implementation and Development

This section explains the implementation process in MATLAB. For more details of MATLAB functions, refer to Appendix A. For all 4 cases, the implementation process is identical. The basic set up for all case is as follows:

1. Load all three camera recordings. This extracts the frames of all three videos.
2. Once all the frames are extracted, convert the data into double-precision numbers for the further mathematical process.
3. Convert all the frames in each recording to grayscale. This is to better locate the light attached at the top of the mass. Since the light will be depicted as solid white, it will have the highest number in grayscale. Finding the maximum grayscale at each frame will locate where the mass is at.

4. There is one problem to be addressed when locating the mass. The reflection on the whiteboard and the person's belt is often misread as the maximum grayscale distracting the process of locating the light. To address, we crop each frame showing only the mass. We set the grayscale of any area outside the motion of the mass as zero, which represents black. This is done by manually checking on each recording. After, we are left with only the portion of the frame that focuses on the mass and its path. Snapshots of cropped frames for all cases are delivered in Appendix C.
5. Find the maximum value and its index inside the grayscale matrix. The index of maximum value will specify the x and y position of the maximum grayscale, the light. Save the x and y position data in vector form for all recordings.
6. Repeat steps 3-5 on all three recordings. However, the videos are not synchronized. Hence, we set the starting point to be where the position for each recording is at its minimum. Then truncate all the position data to the same length. This leaves with synchronized position data for all recordings.
7. Define the data matrix with each row being the x and y position vectors from the three recordings. Find the mean of each row and subtract the mean to the data matrix. This centers the mean of position data to be zero.
8. Perform SVD to get U,Σ, and V of data matrix X.
9. Plot the energy of singular values and determine the necessary dimension of the data or the principal components.
10. Plot the corresponding right singular vectors which capture the independent time oscillation of the specific principal components. Product of ΣV^* gives the projection of z-motion of the mass.

4 Computational Results

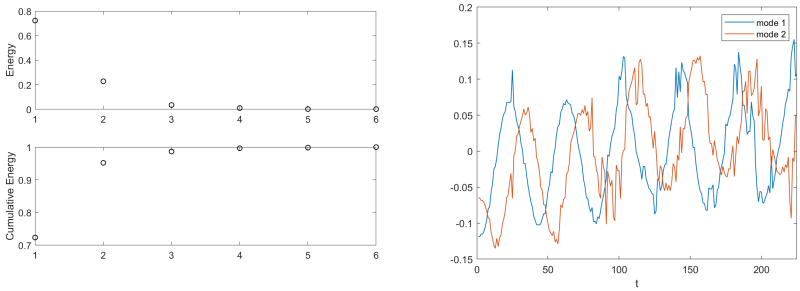
This section discusses the results derived after implementing and developing MATLAB.

4.1 Part 1: Ideal Case

Part 1 of the given task illustrates the ideal case of the bouncing up and down motion of the mass. Thus, the result for this case will very much resemble the theoretical motion. Figure 1 shows the overall computation result for part 1. Notice in figure 1a, the first two singular values have the greatest energy. By the rule of thumb, the modes that add the cumulative sum of energies to 95 percent will be considered as the principal components that we reduce our data dimension to. So we conclude that the actual data dimension is 2. This is essentially true from the eq 3. Since the z-direction motion is modeled by the second-order differential equation, there must be 2 solutions which in our case is shown as the first two modes of PCA. To add on, figure 1b shows the plot of V columns of the first two modes. We can clearly see the oscillations and they represent each of the cosine and sine functions in eq 4. Thus, we conclude that performing PCA very well captures the z-direction motion of the mass.

4.2 Part 2: Noisy Case

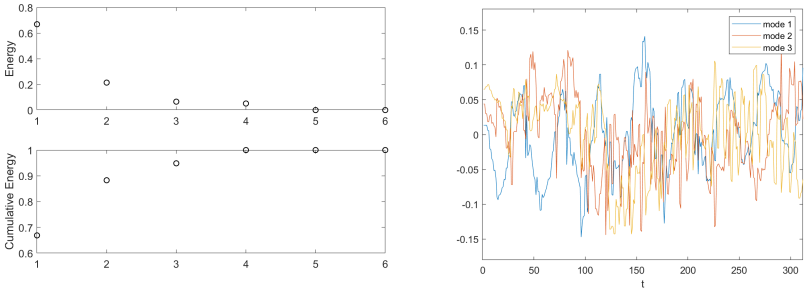
Part 2, the noisy case repeats the ideal case, but introducing the camera shakes as well. Shaking the cameras creates noise in the recordings, therefore, we anticipate a decrease in the accuracy of our PCA model. Figure 2 shows the computational result for Part 2. One limitation that we encountered, in this case, was that the second recording had significantly large noise which made the frame cropping window to be much wider, refer to figure 7. This substantially decreased the accuracy in locating the light attached to the mass. We conjecture that the belt buckle and whiteboard reflection are misread as the brightest points. Likewise, we can see in figure 2a that the energy levels have decreased compared to the ideal case, needing three modes to model the mass motion. Figure 2b validates the impact of decreased accuracy. Although we can see the cosine wave of the first mode, the motion of the second and third modes are very choppy and we can barely see the oscillations. Thus, we conclude that noise has a great effect on decreasing the accuracy of data and the ability to model the motion.



(a) Comparison of energies, and cumulative energies over 6 modes

(b) Plot of columns of V corresponding to the first two modes

Figure 1: Computational Result for the Ideal case



(a) Comparison of energies, and cumulative energies over 6 modes

(b) Plot of columns of V corresponding to the first three modes

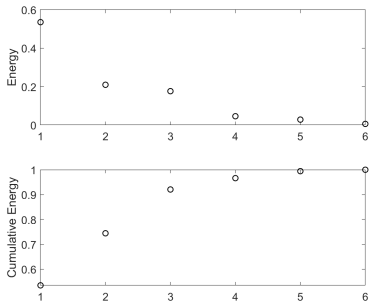
Figure 2: Computational Result for the Noisy case

4.3 Part3: Horizontal Displacement Case

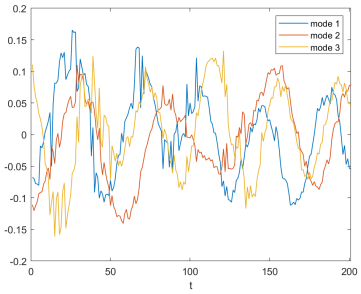
The Horizontal Displacement Case manually shakes the mass off-center to produce both x, y, and z-direction motion. Figure 3 shows the computational results of part 3. From figure 3a, the cumulative energy of the first three modes is around 95 percent. Hence, we plot the first three columns of V to capture the motion of the mass as shown in figure 3b. We can see the cosine waves of oscillation in the z-direction. Further, as time passes, the oscillation is decreasing in size for all modes. We conjecture this is because the mass is moving in the x and y direction at the same time, so the z-direction motion might be relatively shallower than the first two cases. To recapitulate the main result, we need one more principal component mode to capture all x,y, and z-direction motion of the mass unlike the result from part 1 where we only needed two modes to capture the vertical motion.

4.4 Part4: Horizontal Displacement with Rotation Case

The last part of the assignment is to track the motion of mass moving in all x,y, z-direction as well as rotating about the axis. The limitation, in this case, was again locating the light attached to the mass. As the mass rotates, the light also fades. Hence, finding the maximum of the grayscale often located different points in the frame. As a result, the accuracy is significantly lost. Refer to figure 4a. We now need 4 modes to capture the motion of the mass. This is solidified by the figure 4b showing a very choppy plot. We can see the first two modes show the sinusoidal oscillations, but the last two modes capture a lot of noise. Further, the PCA only captured the oscillatory motion, but not the rotation of the mass. We figure this is because the x and y position data don't tell if the mass is rotating or not. Comparing the results altogether, we see that the plots for cases 1 and 3 are relatively clean whereas the plots for cases 2 and 4 are much noisier. Likewise, the effect of noise and locating the object clearly are critical in capturing the motion using PCA.

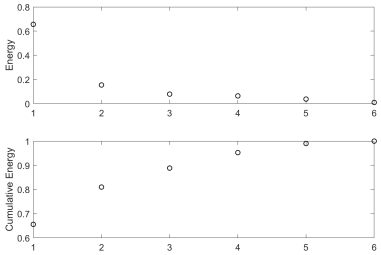


(a) Comparison of energies, and cumulative energies over 6 modes

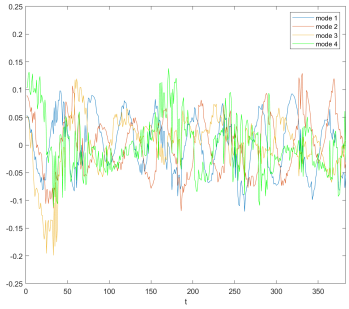


(b) Plot of columns of V corresponding to the first three modes

Figure 3: Computational Result for the Noisy case



(a) Comparison of energies, and cumulative energies over 6 modes



(b) Plot of columns of V corresponding to the first three modes

Figure 4: Computational Result for the Noisy case

5 Summary and Conclusions

In this assignment, we performed Principal Component Analysis to re-express the vertical motion of a mass with only the x and y position information from recordings. Within different cases, we took account of the effects of noise and variations of the situation on modeling the motion. Using PCA, we successfully captured the most meaningful information about the motion and were able to illustrate the oscillatory model of the motion. Further, we compared the results from different cases and concluded that clarity of the motion is critical in obtaining an accurate model. Although there are limitations in PCA with capturing the motion perfectly, it still very well reduces the redundancy and enhances the efficiency greatly when applied to large data sets.

Appendix A MATLAB Functions

This section lists all the important MATLAB Functions used in the process of submarine tracking.

- `abs(X)`: returns the absolute value of each element in array `X`.
- `diag(A)`: returns the diagonal entries in matrix `A`.
- `frame2im(F)`: converts the movie frame F to image data. This is used to convert every frames in recordings to crop the image leaving only the portion of the mass motion.
- `[row,col] = ind2sub(sz,ind)`: converts linear indices to corresponding subscripts of matrix of size `sz`. `row` and `col` in this assignment will be the x and y position of the mass.
- `length(X)`: returns the length of an array `X`.
- `max(X)`: returns the maximum value in array `X`.
- `min(X)`: returns the minimum value in array `X`.
- `mean(A,2)`: returns a column vector of the average values in each row of matrix `A`.
- `repmat(X,m,n)`: creates a `m` by `n` matrix with the values in array `X`. This is used to zero center the data matrix by subtracting the mean of each row.
- `I = rgb2gray(RGB)`: returns the converted grayscale image I from true color RGB image. This is used to better detect the white light attached on the mass. White has the largest grayscale value.
- `size(A)`: returns the dimension of matrix `A`.
- `[U,S,V] = svd(A, 'econ')`: performs economy singular value decomposition on matrix `A`. Returns the U,Σ,V matrices.

Appendix B MATLAB Code

```
clear; close all; clc;
%% Part 1: Ideal

load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')
% implay(vidFrames3_1)

nF1_1 = size(vidFrames1_1,4);
nF2_1 = size(vidFrames2_1,4);
nF3_1 = size(vidFrames3_1,4);

for j = 1:nF1_1
    cam1(j).cdata = vidFrames1_1(:,:,:,j);
    cam1(j).colormap = [];
end

for j = 1:nF2_1
    cam2(j).cdata = vidFrames2_1(:,:,:,j);
    cam2(j).colormap = [];
end

for j = 1:nF3_1
    cam3(j).cdata = vidFrames3_1(:,:,:,j);
    cam3(j).colormap = [];
end

%% Change to grayscale
% cam1
pos1 = [];
for j = 1:nF1_1
    X1gr = rgb2gray(frame2im(cam1(j)));
    %filter can
     X1gr(:,1:310) = 0;
     X1gr(:,390:end) = 0;
     X1gr(1:200,:) = 0;
     X1gr(400:end,:) = 0;
    [mx, ind] = max(X1gr(:));
    [x1, y1] = ind2sub(size(X1gr), ind);
     pos1 = [pos1; mean(x1), mean(y1)];
end

% cam2
pos2 = [];
for j = 1:nF2_1
    X2gr = rgb2gray(frame2im(cam2(j)));
    %filter can
    X2gr(:,1:250) = 0;
    X2gr(:,330:end) = 0;
    X2gr(1:90,:) = 0;
    X2gr(380:end,:) = 0;
```

```
    [mx, ind] = max(X2gr(:));
    [x2, y2] = ind2sub(size(X2gr), ind);
     pos2 = [pos2; x2, y2];
end

% cam 3
pos3 = [];
for j = 1:nF3_1
    X3gr = rgb2gray(frame2im(cam3(j)));
    %filter can
    X3gr(:,1:260) = 0;
    X3gr(:,490:end) = 0;
    X3gr(1:240,:) = 0;
    X3gr(330:end,:) = 0;
     [mx, ind] = max(X3gr(:));
    [x3, y3] = ind2sub(size(X3gr), ind);
     pos3 = [pos3; mean(x3), mean(y3)];
end

%% trimming
[m ind] = min(pos1(1:20,2));
pos1 = pos1(ind:end,:);
[m ind] = min(pos2(1:20,2));
pos2 = pos2(ind:end,:);
[m ind] = min(pos3(1:20,2));
pos3 = pos3(ind:end,:);
pos2 = pos2(1:length(pos3),:); pos1 = pos1(1:length(pos3),:);

% data matrix X
X = [pos1'; pos2'; pos3'];
n = length(pos1); ave = mean(X,2);
X = X-repmat(ave,1,n);
[U,S,V] = svd(X, 'econ');
sig = diag(S);

%% plotting
figure()
subplot(3,1,1)
plot(sig, 'ko','Linewidth',1)
ylabel('\sigma')
set(gca,'Fontsize',12,'Xtick',0:1:6)
subplot(3,1,2)
plot(sig.^2/sum(sig.^2),'ko','Linewidth',1)
ylabel('Energy')
set(gca,'Fontsize',12,'Xtick',0:1:6)
subplot(3,1,3)
plot(cumsum(sig.^2)/sum(sig.^2),'ko','Linewidth',1)
ylabel('Cumulative Energy')
set(gca,'Fontsize',12,'Xtick',0:1:6)

figure()
plot(V(:,1), 'LineWidth', .8)
hold on
plot(V(:,2), 'LineWidth', .8)
```

```
xlim([0 225])
xlabel('t');  legend('mode 1', 'mode 2');


%%
clear; close all; clc;
%% Case 2: Noisy
load('cam1_2.mat')
load('cam2_2.mat')
load('cam3_2.mat')

nF1_2 = size(vidFrames1_2,4);
nF2_2 = size(vidFrames2_2,4); % max size
nF3_2 = size(vidFrames3_2,4);

for j = 1:nF2_2
    if j <= nF1_2
        cam1(j).cdata = vidFrames1_2(:,:,:,j);
        cam1(j).colormap = [];
    end
    if j <= nF3_2
        cam3(j).cdata = vidFrames3_2(:,:,:,j);
        cam3(j).colormap = [];
    end
    cam2(j).cdata = vidFrames2_2(:,:,:,j);
    cam2(j).colormap = [];
end

%%
pos1 = []; pos2 = []; pos3 = [];
for j = 1:nF2_2
    if j <= nF1_2
        X1gr = rgb2gray(frame2im(cam1(j)));
        %filter can
        X1gr(:,1:310) = 0;
        X1gr(:,400:end) = 0;
        X1gr(1:220,:) = 0;
        X1gr(370:end,:) = 0;
        [mx, ind] = max(X1gr(:));
        [x1, y1] = ind2sub(size(X1gr), ind);
        pos1 = [pos1; x1, y1];
    end
    if j <= nF3_2
        X3gr = rgb2gray(frame2im(cam3(j)));
        %filter can
        X3gr(:,1:285) = 0;
        X3gr(:,450:end) = 0;
        X3gr(1:205,:) = 0;
        X3gr(300:end,:) = 0;
        [mx, ind] = max(X3gr(:));
        [x3, y3] = ind2sub(size(X3gr), ind);
        pos3 = [pos3; x3, y3];
    end
    X2gr = rgb2gray(frame2im(cam2(j)));
```

```
    %filter can
    X2gr(:,1:220) = 0;
    X2gr(:,400:end) = 0;
    X2gr(1:50,:) = 0;
    X2gr(380:end,:) = 0;
    [mx, ind] = max(X2gr(:));
    [x2, y2] = ind2sub(size(X2gr), ind);
     pos2 = [pos2; x2, y2];
end
%%
[m ind] = min(pos1(1:20,2));
pos1 = pos1(ind:end,:);
[m ind] = min(pos2(1:20,2));
pos2 = pos2(ind:end,:);
[m ind] = min(pos3(1:20,2));
pos3 = pos3(ind:end,:);
pos2 = pos2(1:length(pos1), :); pos3 = pos3(1:length(pos1), :);

X = [pos1'; pos2'; pos3'];
n = length(pos1);
ave = mean(X,2);
X = X-repmat(ave,1,n);
[U,S,V] = svd(X, 'econ');
sig = diag(S);

%% plotting
figure()
subplot(3,1,1)
plot(sig, 'ko','Linewidth',1)
ylabel('\sigma')
set(gca,'Fontsize',12,'Xtick',0:1:6)
subplot(3,1,2)
plot(sig.^2/sum(sig.^2),'ko','Linewidth',1)
ylabel('Energy')
set(gca,'Fontsize',12,'Xtick',0:1:6)
subplot(3,1,3)
plot(cumsum(sig.^2)/sum(sig.^2),'ko','Linewidth',1)
ylabel('Cumulative Energy')
set(gca,'Fontsize',12,'Xtick',0:1:6)

figure()
plot(V(:,1), 'LineWidth', .5)
hold on
plot(V(:,2), 'LineWidth', .5)
hold on
plot(V(:,3), 'LineWidth', .5)
axis([0 312 -0.18 0.18])
xlabel('t');  legend('mode 1', 'mode 2', 'mode 3');


%%
clear; close all; clc;
%% Part 3: Horizontal Displacement
load('cam1_3.mat')
```

```
load('cam2_3.mat')
load('cam3_3.mat')

nF1_3 = size(vidFrames1_3,4);
nF2_3 = size(vidFrames2_3,4); % max size
nF3_3 = size(vidFrames3_3,4);

for j = 1:nF2_3
    if j <= nF1_3
        cam1(j).cdata = vidFrames1_3(:,:,:,j);
        cam1(j).colormap = [];
    end
    if j <= nF3_3
        cam3(j).cdata = vidFrames3_3(:,:,:,j);
        cam3(j).colormap = [];
    end
    cam2(j).cdata = vidFrames2_3(:,:,:,j);
    cam2(j).colormap = [];
end

pos1 = []; pos2 = []; pos3 = [];
for j = 1:nF2_3
    if j <= nF1_3
        X1gr = rgb2gray(frame2im(cam1(j)));
        %filter can
        X1gr(:,1:250) = 0;
        X1gr(:,410:end) = 0;
        X1gr(1:200,:) = 0;
        X1gr(400:end,:) = 0;
        [mx, ind] = max(X1gr(:));
        [x1, y1] = ind2sub(size(X1gr), ind);
        pos1 = [pos1; x1, y1];
    end
    if j <= nF3_3
        X3gr = rgb2gray(frame2im(cam3(j)));
        %filter can
        X3gr(:,1:270) = 0;
        X3gr(:,490:end) = 0;
        X3gr(1:150,:) = 0;
        X3gr(330:end,:) = 0;
        [mx, ind] = max(X3gr(:));
        [x3, y3] = ind2sub(size(X3gr), ind);
        pos3 = [pos3; x3, y3];
    end
    X2gr = rgb2gray(frame2im(cam2(j)));
    %filter can
    X2gr(:,1:200) = 0;
    X2gr(:,400:end) = 0;
    X2gr(1:150,:) = 0;
    X2gr(400:end,:) = 0;
    [mx, ind] = max(X2gr(:));
    [x2, y2] = ind2sub(size(X2gr), ind);
     pos2 = [pos2; x2, y2];
end
```

```
%%
[m ind] = min(pos1(1:20,2));
pos1 = pos1(ind:end,:);
[m ind] = min(pos2(1:20,2));
pos2 = pos2(ind:end,:);
[m ind] = min(pos3(1:20,2));
pos3 = pos3(ind:end,:);
pos1 = pos1(1:length(pos3),:); pos2 = pos2(1:length(pos3),:);

X = [pos1'; pos2'; pos3'];
n = length(pos1);
ave = mean(X,2);
X = X-repmat(ave,1,n);
[U,S,V] = svd(X, 'econ');
sig = diag(S);

%% plotting
figure()
subplot(3,1,1)
plot(sig, 'ko','Linewidth',1)
ylabel('\sigma')
set(gca,'Fontsize',12,'Xtick',0:1:6)
subplot(3,1,2)
plot(sig.^2/sum(sig.^2),'ko','Linewidth',1)
ylabel('Energy')
set(gca,'Fontsize',12,'Xtick',0:1:6)
subplot(3,1,3)
plot(cumsum(sig.^2)/sum(sig.^2),'ko','Linewidth',1)
ylabel('Cumulative Energy')
set(gca,'Fontsize',12,'Xtick',0:1:6)

figure()
plot(V(:,1), 'LineWidth', .8)
hold on
plot(V(:,2), 'LineWidth', .8)
hold on
plot(V(:,3), 'LineWidth', .8)
xlim([0 201])
xlabel('t');  legend('mode 1', 'mode 2', 'mode 3');


%%
clear; close all; clc;
%% Part 4: Horizontal Displacement and Rotation
load('cam1_4.mat')
load('cam2_4.mat')
load('cam3_4.mat')

nF1_4 = size(vidFrames1_4,4);
nF2_4 = size(vidFrames2_4,4); % max size
nF3_4 = size(vidFrames3_4,4);

for j = 1:nF2_4
```

```
    if j <= nF1_4
        cam1(j).cdata = vidFrames1_4(:,:,:,j);
        cam1(j).colormap = [];
    end
    if j <= nF3_4
        cam3(j).cdata = vidFrames3_4(:,:,:,j);
        cam3(j).colormap = [];
    end
    cam2(j).cdata = vidFrames2_4(:,:,:,j);
    cam2(j).colormap = [];
end
%%
pos1 = []; pos2 = []; pos3 = [];
for j = 1:nF2_4
    if j <= nF1_4
        X1gr = rgb2gray(frame2im(cam1(j)));
        %filter can
        X1gr(:,1:320) = 0;
        X1gr(:,450:end) = 0;
        X1gr(1:230,:) = 0;
        X1gr(380:end,:) = 0;
        [m1, ind1] = max(X1gr(:));
        [x1, y1] = ind2sub(size(X1gr), ind1);
        pos1 = [pos1; x1, y1];
    end
    if j <= nF3_4
        X3gr = rgb2gray(frame2im(cam3(j)));
        %filter can
        X3gr(:,1:300) = 0;
        X3gr(:,470:end) = 0;
        X3gr(1:160,:) = 0;
        X3gr(270:end,:) = 0;
        [m3, ind3] = max(X3gr(:));
        [x3, y3] = ind2sub(size(X3gr), ind3);
        pos3 = [pos3; x3, y3];
    end
    X2gr = rgb2gray(frame2im(cam2(j)));
    %filter can
    X2gr(:,1:230) = 0;
    X2gr(:,410:end) = 0;
    X2gr(1:95,:) = 0;
    X2gr(330:end,:) = 0;
    [m2, ind2] = max(X2gr(:));
    [x2, y2] = ind2sub(size(X2gr), ind2);
     pos2 = [pos2; x2, y2];
end

%%
[m ind] = min(pos1(1:20,2));
pos1 = pos1(ind:end,:);
[m ind] = min(pos2(1:20,2));
pos2 = pos2(ind:end,:);
[m ind] = min(pos3(1:20,2));
pos3 = pos3(ind:end,:);
```

```
pos2 = pos2(1:length(pos3),:); pos1 = pos1(1:length(pos3),:);

X = [pos1'; pos2'; pos3'];
n = length(pos1);
ave = mean(X,2);
X = X-repmat(ave,1,n);
[U,S,V] = svd(X, 'econ');
sig = diag(S);

%% plotting
figure()
subplot(3,1,1)
plot(sig, 'ko','Linewidth',1)
ylabel('\sigma')
set(gca,'Fontsize',12,'Xtick',0:1:6)
subplot(3,1,2)
plot(sig.^2/sum(sig.^2),'ko','Linewidth',1)
ylabel('Energy')
set(gca,'Fontsize',12,'Xtick',0:1:6)
subplot(3,1,3)
plot(cumsum(sig.^2)/sum(sig.^2),'ko','Linewidth',1)
ylabel('Cumulative Energy')
set(gca,'Fontsize',12,'Xtick',0:1:6)

figure()
plot(V(:,1), 'LineWidth', .5)
hold on
plot(V(:,2), 'LineWidth', .5)
hold on
plot(V(:,3), 'LineWidth', .5)
hold on
plot(V(:,4), 'c', 'LineWidth', .5)
axis([0 383 -0.25 0.25])
xlabel('t');  legend('mode 1', 'mode 2', 'mode 3', 'mode 4');
```

Appendix C Supplementary Figures

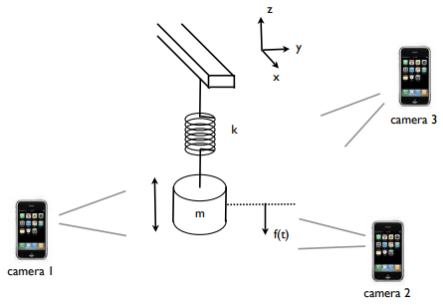
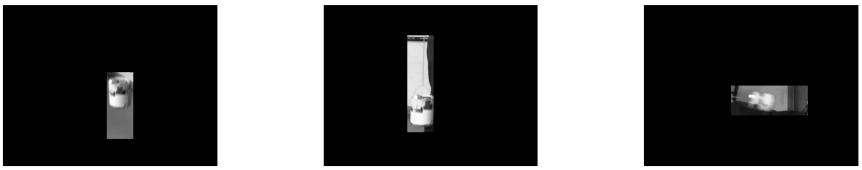


Figure 5: Structure of the assignment setup (Kutz)



(a) Camera 1 (b) Camera 2 (c) Camera 3

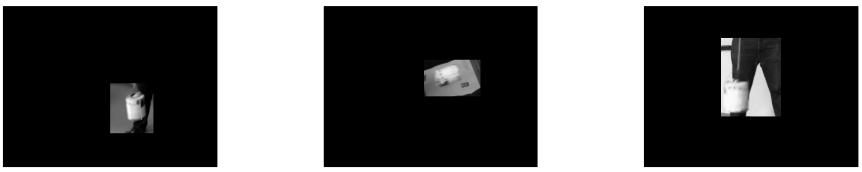
Figure 6: Cropped frame for Ideal Case

(a) Camera 1 (b) Camera 2 (c) Camera 3

Figure 7: Cropped frame for Noisy Case

(a) Camera 1 (b) Camera 2 (c) Camera 3

Figure 8: Cropped frame for Horizontal Displacement Case



(a) Camera 1 (b) Camera 2 (c) Camera 3

Figure 9: Cropped frame for Horizontal Displacement with Rotation Case