AMATH 482 HW2: PCA

Ken Lo

February 11, 2018

Abstract

Use Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) to examine how noise and complexity of the data affects the performance of PCA. The data is obtained by extracting position of a paint can in a spring-mass system, with different noise or complexity introduced in different tests.

1 Introduction and Overview

Principal Component Analysis (PCA) is a method to reduce redundancy in data such that we can represent the data in lower dimensions. However, real-world data is not always ideal, they can be full of noises and corruptions. And such imperfections of our data can negatively impact our analysis results.

In this assignment, we will examine how noises and more complex data can impact PCA. We will extract locations of a paint can from 12 videos(4 scenariors, 3 video each), and perform SVD on these data to see how our results may differ in these different ideal and non-ideal scenarios.

The first case is the ideal case, where the paint can is in simple harmonic motion; the second case is the noisy case, where the videos shakes while recording the simple harmonic motion; the third case introduces a horizontal displacement(pendulum motion) of the paint can, in addition to the simple harmonic motion; the last case introduces an additional rotation to the paint can, with the pendulum and simple harmonic motion.

2 Theoretical Background

2.1 Singular Value Decomposition

A Singular Value Decomposition (SVD) takes the form

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$$

where

$$\mathbf{U} \in \mathbb{C}^{m\times m} \text{is unitary}$$
$$\mathbf{V} \in \mathbb{C}^{n\times n} \text{is unitary}$$
$$\mathbf{\Sigma} \in \mathbb{R}^{m\times n} \text{is diagonal}$$

Here the matrix $\mathbf{\Sigma}$ is diagonal with entries $\sigma_1 < \sigma_2 < \cdots < \sigma_n$.

By theorem 14.1.77 from class notes, any matrix $\mathbf{A}$ always has an SVD. So we can use this fact to perform SVD on our data matrix, in which the rows are matrices with x

or y coordinate of the paint can. By doing so, our data matrix $\mathbf{A}$ will be diagonalized using basis $\mathbf{U}$ and $\mathbf{V}$, with singular values in $\mathbf{\Sigma}$. Suppose redundancy is present among our measurements captured from different cameras, $\mathbf{\Sigma}$ will only have a few large singular values, with the rest of the singular values being very close to 0. The most prominent modes of our spring-mass dynamics will take place as vectors in $\mathbf{U}$.

2.2 Principal Component Analysis (PCA)

Suppose we have two sets of data, in vectors $\mathbf{a} = [a_1 \cdots a_n]$ and $\mathbf{b} = [b_1 \cdots b_n]$ then the covariance between these two different datasets is given by

$$\sigma_{ab}^2 = \frac{1}{n-1}\mathbf{a}\mathbf{b}^T$$

. We can generalize the above expression with a *covariance matrix*

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

. The $m \times m$ covariance matrix $\mathbf{C_X}$ has diagonals representing the variance of each measurement, with the off-diagonals representing the covariance between measurement types. Therefore, the smaller the off-diagonal entries, the smaller the statistical independence between two measured quantites, and vice versa.

Our goal with the covariance matrix is to have $\mathbf{C_X}$ contains diagonals ordered from largest to smallest with off-diagonals equal to zero. In other words, we want to diagonalize the covariance matrix. The goal of PCA is to find a basis in which the covariance matrix is diagonal. And we will achieve this by using Singular Value Desomposition.

Through SVD, we will obtain a pair of bases $\mathbf{U}$ and $\mathbf{V}$. Then we can introduce a new transformed variable

$$\mathbf{Y} = \mathbf{U}^*\mathbf{X}$$

where $\mathbf{X}$ is our data matrix. Then we can compute the variance in $\mathbf{Y}$:

$$\begin{aligned}
\mathbf{C_Y} &= \frac{1}{n-1}\mathbf{Y}\mathbf{Y}^T \\
&= \frac{1}{n-1}(\mathbf{U}^*\mathbf{X})(\mathbf{U}^*\mathbf{X})^T \\
&= \frac{1}{n-1}\mathbf{U}^*(\mathbf{X}\mathbf{X}^T)\mathbf{U} \\
&= \frac{1}{n-1}\mathbf{U}^*\mathbf{U}\mathbf{\Sigma^2}\mathbf{U}\mathbf{U}^* \\
\mathbf{C_Y} &= \frac{1}{n-1}\mathbf{\Sigma^2}
\end{aligned}$$

And now we have a diagonalized covariance matrix.

3 Algorithm Implementation and Development

3.1 Data Extraction

To obtain the locations of the paint can, we will use the flashlight placed on top of the can to help us. Since the brightest pixel corresponds to the highest RGB sum, our algorithm will search for a local maximum of RGB sum, within a small square search window.

The search window is a constraint we place when searching for the brightest pixel. Since each frame might have other points which are brighter than the flashlight(such as the white board), we will restrict our search to be within ± 20 pixels from the previous brightest location. To achieve this, we also need to define(by manually clicking on the image) the initial location of the flashlight, which is done by using the `ginput` command.

3.2 SVD

After obtaining the coordinates, we normalize the coordinates such that they are in [01]. We also enforce each camera to have the same number of frames, by using only the minimum number of frames of any of the three cameras, and trim away the excess frames in the other two cameras. Prior to performing the SVD, we set the mean and variance of each row of our data matrix to be 0 and 1, respectively.

After the above steps, we perform SVD on these data.

4 Computational Results

4.1 Test 1: Ideal Case

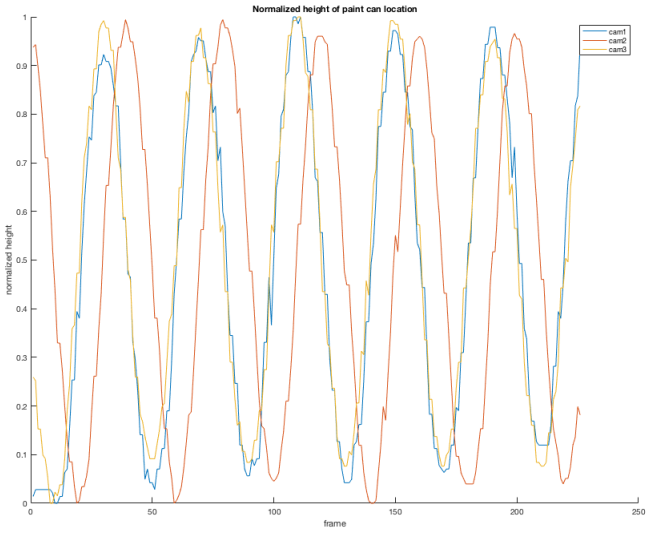

Figure 1: Normalized height over each frame measured in each camera in test 1

From figure 1, we can see the three cameras both recorded the same oscillatory signal, despite minor differences in their starting positions and amplitutes. So, we can say that there is a lot of redundancy in our data.

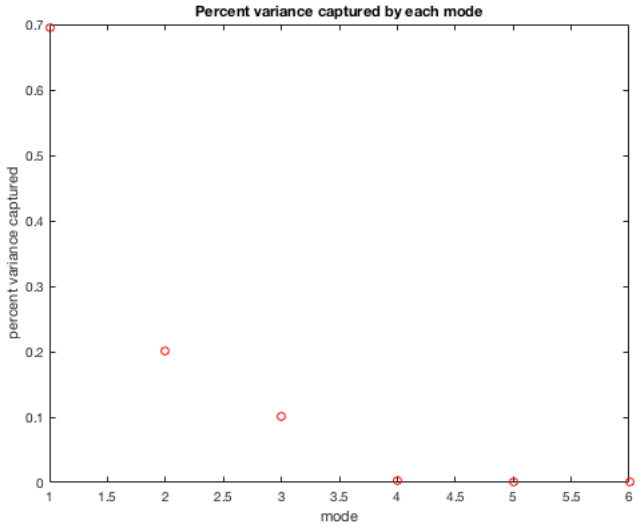

Figure 2: Percent variance captured by each mode

In figure 2, we see the first mode captured about 70% of the variances, and the second and third mode captured the remaining 20% and 10%, respectively.

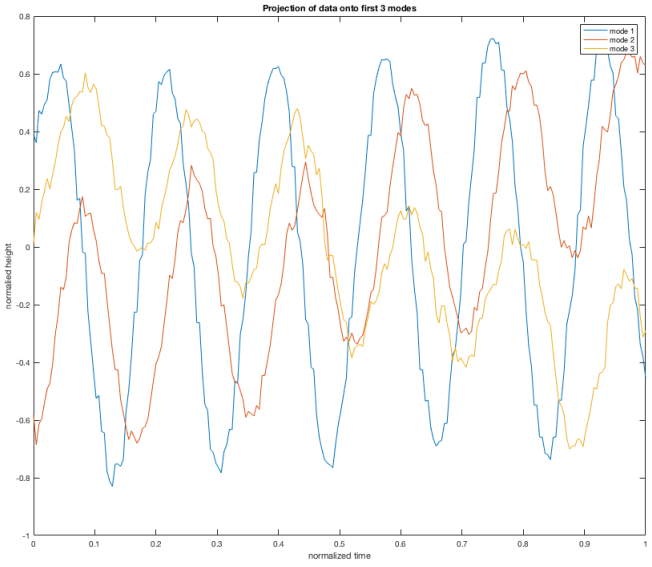

Figure 3: Projection of data onto the first 3 modes

Figure 3 showed how the data projected onto the first three, most prominent modes. Here we see that the first mode(blue line) showed the oscillatory behavior of the spring-mass system. While the second and third mode also showed some oscialltions. This could be attributed to the fact that the oscillations did not all start at the first frame for all three cameras, which we can also see from 1: camera 2 slightly lagged behind cameras 1 and 3. We can also see that the third mode showed a decrease of height over time which can be caused by the damping effect of the spring-mass system.

4.2 Test 2: Noisy Case

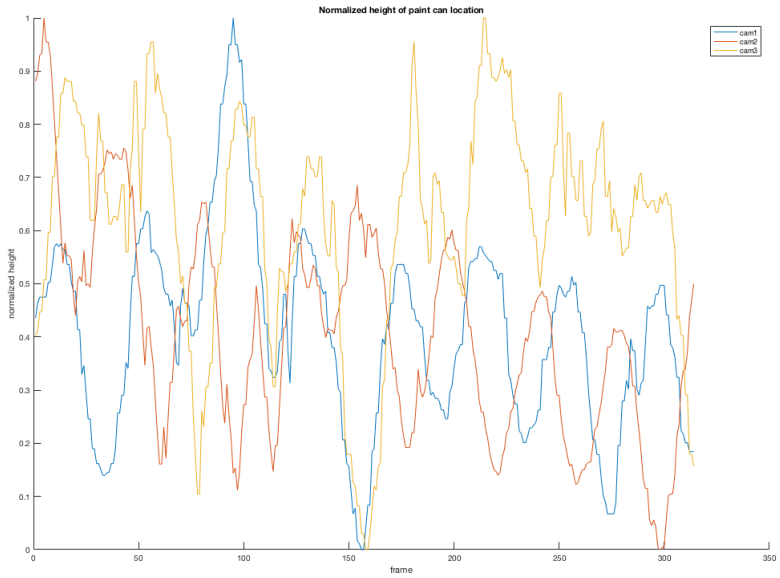

Figure 4: Normalized height over each frame measured in each camera in test 2

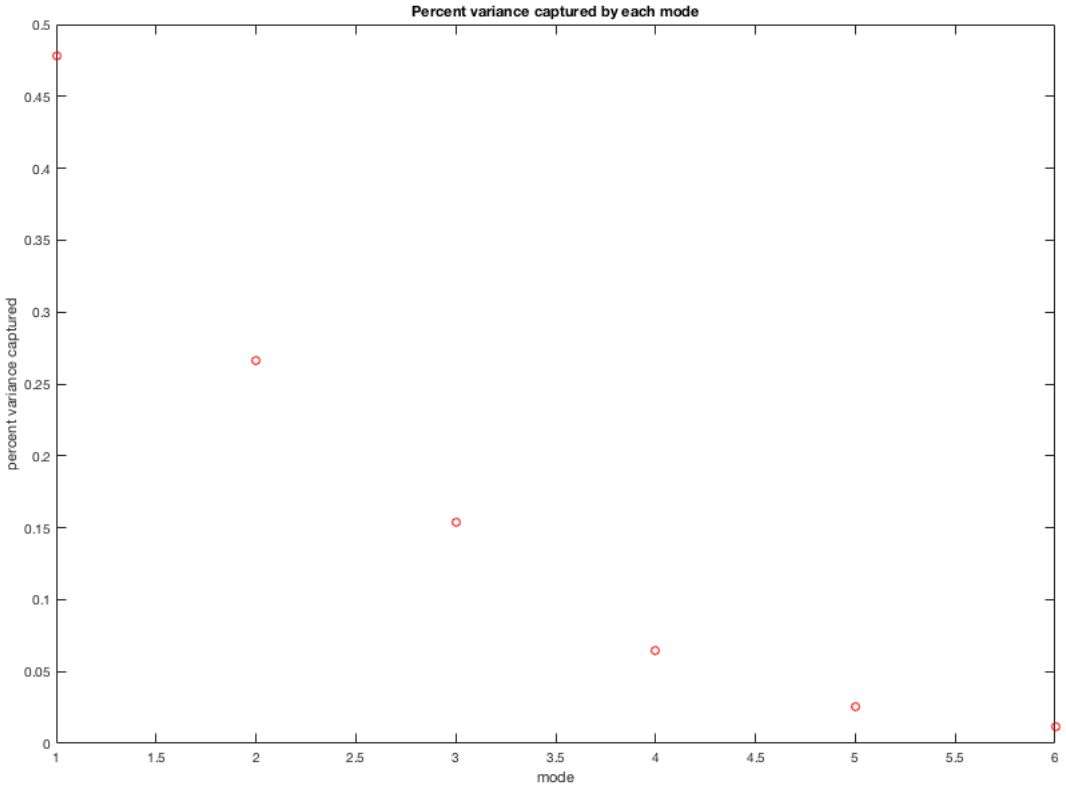

Figure 5: Percent variance captured by each mode

Figure 6: Projection of data onto the first 6 modes

In figure 4, we can barely tell the paint can is in an harmonic oscillatory motion with the amount of noise presented in the data. As seen in figure 5, the variance is distributed more evenly across each mode, since SVD could not reduce much of the redundnacy due to the noise. Projecting the data onto those 6 modes in figure 6, we could hardly tell there is any oscillatory behavior in our data.

4.3 Test 3: Horizontal Displacement

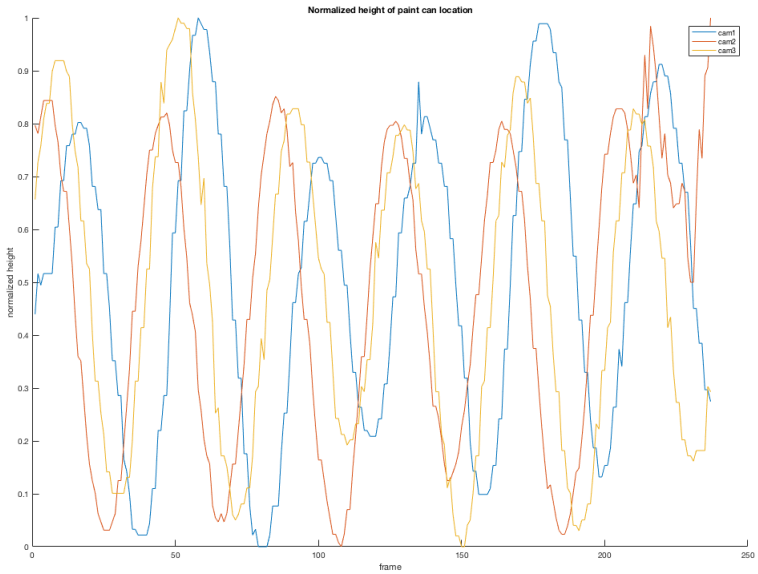

Figure 7: Normalized height over each frame measured in each camera in test 3

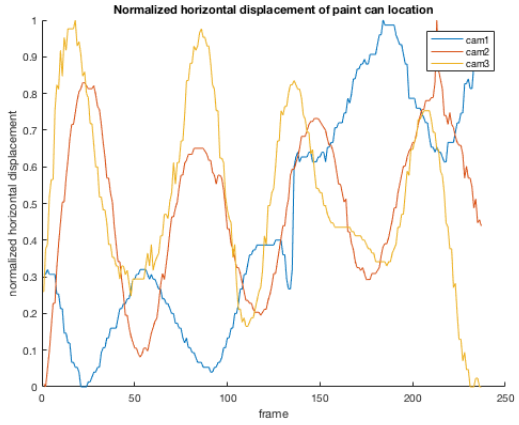

Figure 8: Normalized horizontal displacement over each frame measured in each camera in test 3

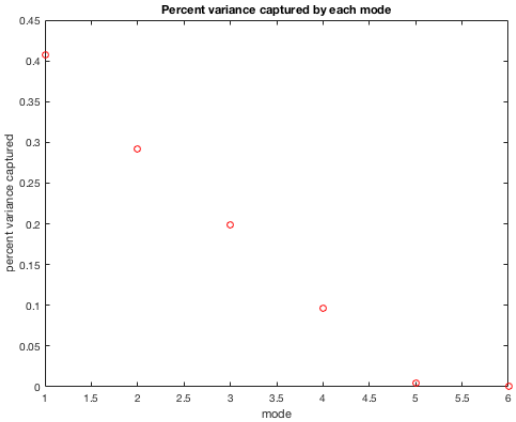

Figure 9: Percent variance captured by each mode

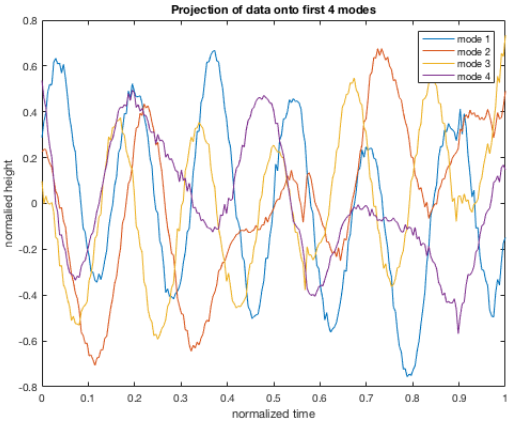

Figure 10: Projection of data onto the first 4 modes

In this case, the paint can had both a pendulum motion and a simple harmonic motion. Other than the harmonic motion in observed in figure 7, we can also see a periodic motion exhibited through figure 8. In figure 9, we see that nearly all the variance is captured by the first 4 modes. Our projection of data onto the first 2 modes in figure 10 both show

a clear oscillatory motion, which could represent the simple harmonic motion and the pendulum motion respectively. Modes 3 and 4 also showed some oscillatory behaviors.

4.4 Test 4: Horizontal Displacement and Rotation

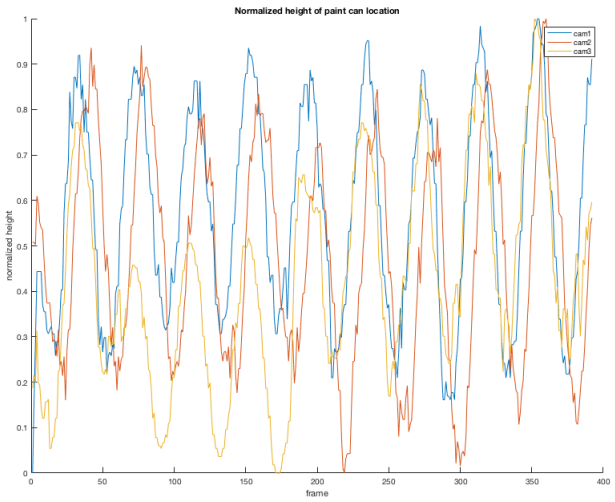

Figure 11: Normalized height over each frame measured in each camera in test 4

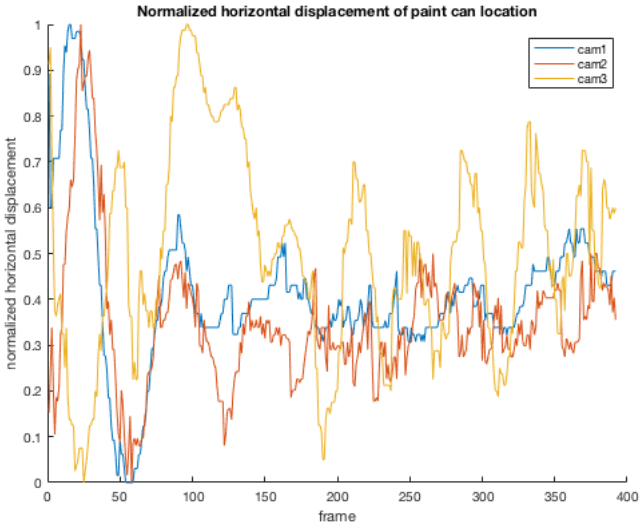

Figure 12: Normalized horizontal displacement over each frame measured in each camera in test 4

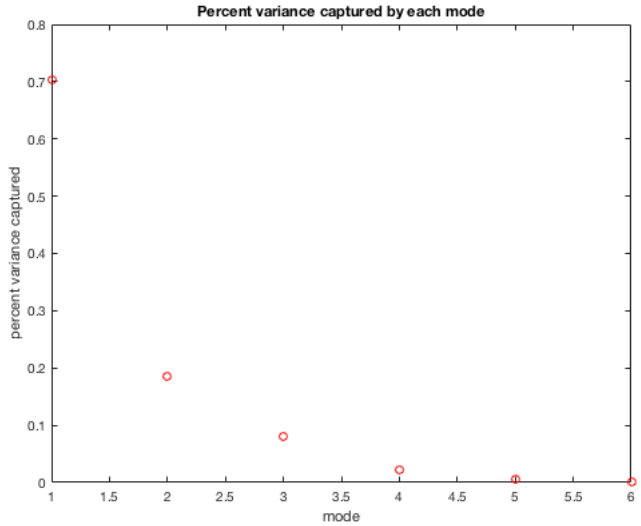

Figure 13: Percent variance captured by each mode

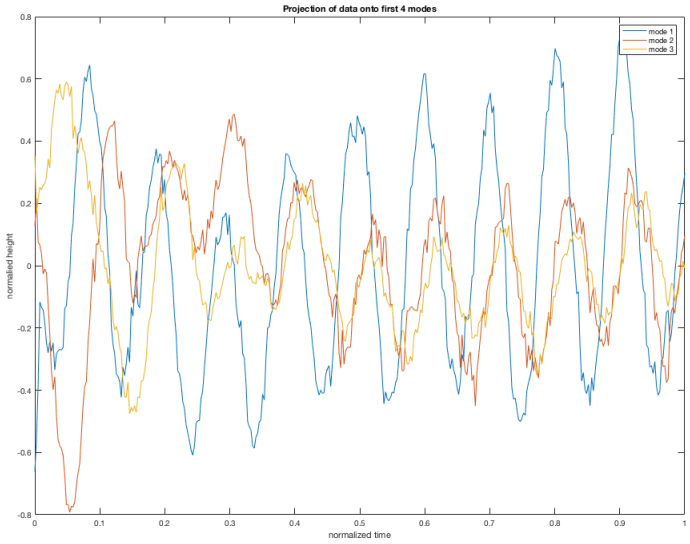

Figure 14: Projection of data onto the first 3 modes

In figure 11, the measured height of the paint can resembles a more noisy version of 1. However, even with a horizontal displacement just like test 3, 12 did not evidently show the horizontal pendulum motion. Figure 13 shows that nearly all the variance is captured by the first 3 modes, which we project our data on in figure 14. In this figure, we see the first mode shows an oscillatory behavior just as in test 1 and 3. The second and third modes are also somewhat periodic, and also resemble modes 2 and 3 in test 1(see figure 3).

5 Summary and Conclusions

In this assignment, we used SVD to perform PCA on camera data recording a spring-mass system of a paint can. In different test cases with different noises and complexities to our data, we saw how the performance of PCA could be negatively affected by noise, which

is shown in test 2. In test cases 1, 3 and 4, we were also able to represent the simple harmonic motion in one dimension. In addition, PCA also gave us an additional mode to represent the pendulum motion exhibited by the paint can. However, PCA had trouble uncovering the pendulum motion in case 4, potentially due to the rotation making the motion of the paint can harder to detect.

A MATLAB functions used and brief implementation explanation

- zeros() - make a matrix with zeros
- imshow() - show the image represented by a matrix
- ginput() - manually click on the coordinate on a GUI
- sum() - take the sum along an axis
- max() - find the maximum along an axis
- double() - change the array to have type double
- transpose() - transpose a matrix
- repmat() - create a matrix consist of M-by-N tiling of copies of A
- svd() - perform svd
- linspace() - make a linear space in the defined range
- diag() - get the diagonal entries of a matrix

B MATLAB codes

B.1 extract.m

```
% call getcoord() to get locations of the paint can from video frames
clear all; close all; clc
load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')

coords1_1 = getcoord(vidFrames1_1, 20);
coords2_1 = getcoord(vidFrames2_1, 20);
coords3_1 = getcoord(vidFrames3_1, 20);

load('cam1_2.mat')
load('cam2_2.mat')
load('cam3_2.mat')

coords1_2 = getcoord(vidFrames1_2, 20);
coords2_2 = getcoord(vidFrames2_2, 20);
coords3_2 = getcoord(vidFrames3_2, 20);

load('cam1_3.mat')
load('cam2_3.mat')
load('cam3_3.mat')

coords1_3 = getcoord(vidFrames1_3, 20);
```

```
coords2_3 = getcoord(vidFrames2_3, 20);
coords3_3 = getcoord(vidFrames3_3, 20);

load('cam1_4.mat')
load('cam2_4.mat')
load('cam3_4.mat')

coords1_4 = getcoord(vidFrames1_4, 20);
coords2_4 = getcoord(vidFrames2_4, 20);
coords3_4 = getcoord(vidFrames3_4, 20);
```

B.2 getcoord.m

```
% function to get the coordinates from video frame images
function coords = getcoord(frames, wsize)
    coords = zeros(size(frames, 4), 2); % initialize coords array
    imshow(frames(:,:,:,1));
    coords(1,:) = ginput(1); % user input initial coordinate
    coords = uint16(coords);
    for i=2:size(frames, 4)
        prev = coords(i-1,:);
        bot = prev(2) - wsize;
        top = prev(2) + wsize;
        left = prev(1) - wsize;
        right = prev(1) + wsize;
        searchbox = sum(frames(bot:top,left:right,:,i), 3);
        [maxrow, ys] = max(searchbox);
        [~, x] = max(maxrow);
        coords(i,:) = [left + x, bot + ys(x)];
    end
end
```

B.3 normzlize.m

```
% function to normalize a data vector
function [toReturn] = normalize(coords)
    toReturn = zeros(size(coords));
    for i=1:2
        toReturn(:,i) = coords(:,i) - min(coords(:,i));
        toReturn(:,i) = toReturn(:,i) / max(toReturn(:,i));
    end
end
```

B.4 analysis.m

```
    % perform analysis on the coordinates

    %% load coordinates
    clear all;
    load('coordinates.mat');

    %% convert data type to double
```

```
coords1_1 = double(coords1_1);
coords2_1 = double(coords2_1);
coords3_1 = double(coords3_1);
coords1_2 = double(coords1_2);
coords2_2 = double(coords2_2);
coords3_2 = double(coords3_2);
coords1_3 = double(coords1_3);
coords2_3 = double(coords2_3);
coords3_3 = double(coords3_3);
coords1_4 = double(coords1_4);
coords2_4 = double(coords2_4);
coords3_4 = double(coords3_4);

%% normalize data
normalized1_1 = normalize(coords1_1);
normalized2_1 = normalize(coords2_1);
normalized3_1 = normalize(coords3_1);
normalized1_2 = normalize(coords1_2);
normalized2_2 = normalize(coords2_2);
normalized3_2 = normalize(coords3_2);
normalized1_3 = normalize(coords1_3);
normalized2_3 = normalize(coords2_3);
normalized3_3 = normalize(coords3_3);
normalized1_4 = normalize(coords1_4);
normalized2_4 = normalize(coords2_4);
normalized3_4 = normalize(coords3_4);


%% plot normalized height
minn = 392; % min number of frames among the three cameras
normalized1 = normalize(coords1_4(1:minn,:));
normalized2 = normalize(coords2_4(1:minn,:));
normalized3 = normalize(coords3_4(1:minn,:));
figure
hold on
plot(normalized1(:,2));
plot(normalized2(:,2));
plot(normalized3(:,1));
legend("cam1", "cam2", "cam3")
title("Normalized height of paint can location")
xlabel("frame")
ylabel("normalized height")

%% plot horizontal displacement
figure
hold on
plot(normalized1(:,1));
plot(normalized2(:,1));
plot(normalized3(:,2));
legend("cam1", "cam2", "cam3")
title("Normalized horizontal displacement of paint can location")
xlabel("frame")
ylabel("normalized horizontal displacement")
```

```
%% stack x y coordinates and perform svd
X = [transpose(normalized1(1:minn,1));  % stack the coordinates from each camera
        transpose(normalized1(1:minn,2));
        transpose(normalized2(1:minn,1));
        transpose(normalized2(1:minn,2));
        transpose(normalized3(1:minn,1));
        transpose(normalized3(1:minn,2))];


%%
[m,n]=size(X);
mn = mean(X, 2);
X = X-repmat(mn, 1, n);

[u,s,v] = svd(X/sqrt(n-1));
lambda=diag(s).^2; % variance
Y = u'*X;

%% plot variance
figure
plot(diag(lambda)/sum(diag(lambda)), 'ro');
title('Percent variance captured by each mode');
xlabel('mode')
ylabel('percent variance captured')
%% projection of data
t = linspace(0,1,n); % normalize time [0 1]
figure
plot(t, Y(1,:), t, Y(2,:), t, Y(3,:))%, t, Y(4,:))%, t, Y(5,:), t, Y(6,:));
title("Projection of data onto first 4 modes")
xlabel("normalized time")
ylabel("normalied height")
legend("mode 1", "mode 2", "mode 3")% , "mode 4")%, "mode 5", "mode 6")
```