AMATH 582 Homework 3

Shabab Ahmed
Github link: https://github.com/sahmed95/AMATH582

February 21, 2020

Abstract

In this project, we illustrate various aspects of the PCA (Prinicpal Component Analysis) and its usefulness using an experiment involving a spring-mass system. We use three different cameras to extract data concerning the behavior of the system and then we empirically extract the dynamics of the system using PCA. We consider four different cases: 1) simple harmonic motion, 2) simple harmonic motion but with noisy measurements, 3)horizontal displacement and 4)horizontal displacement with rotation. The project involves two parts: 1) tracking the paint can and 2) using the SVD to get the principal modes of the system. We find that the PCA is able to extract the underlying dynamics of the system without the governing equations even with noisy data demonstrating its strength and practicality.

1 Introduction and Overview

Linear algebra plays a crucial role in data analysis and computation. At the heart of the use of linear algebra in data analysis is the Singular Value Decomposition (SVD). SVD is an extremely powerful technique that has a variety of applications. The theoretical aspect of the SVD will be described in the Theoretical Background section. One of the key applications of SVD is the Principal Component Analysis (PCA).

PCA is method to produce low dimensional reductions of the dynamics and behavior of a system when the governing equations are not known from seemingly random data. PCA can be used to transform a set of possibly correlated variables into some more fundamental set of independent variables. In many data sets, where there are lots of variables it is possible that groups of variables move together. It is possible that more than one variable is measuring the same driving principle governing the behavior of the system. PCA generates a new set of variables called the principle components. Each principal component is a linear combination of the original variables and all the principal components are orthogonal, that is, uncorrelated to each other. The principal components are single axes in space. When we project each observation on those axes, the resulting values form new variables. We use SVD to perform PCA on our data set.

In this project we consider a simple spring-mass system. The oscillating mass in our case is a paint can. The paint can is attached to a string and perturbed in different ways. The experiment is carried out for four different cases. In the first case (ideal case) the paint can undergoes simple harmonic motion in the z direction. In the second test, the paint can undergoes simple harmonic motion but noise is introduced into the video by possible shake of the camera. The third case introduces to a slightly different system with the can being released slightly off center. This produces motion in the x-y plane as well as the z direction. The final case considers both horizontal displacement and rotation and so there is both a pendulum motion and a simple harmonic motion. The behavior in each of these cases is recorded using three different cameras. Each camera is in a different position and so records the behavior from different angles. They produce two-dimensional representation of the data. We denote the data from the three cameras with subscripts $1, 2$ and 3. Then, the data collected are represented by the following:

$$\begin{aligned} &\text{camera 1: } (\mathbf{x}_1, \mathbf{y}_1) \\ &\text{camera 2: } (\mathbf{x}_2, \mathbf{y}_2) \\ &\text{camera 3: } (\mathbf{x}_3, \mathbf{y}_3) \end{aligned}$$

One important thing to note is that this is not the same x-y plane of the oscillating can system. Finally,we gather the data into a single matrix to form:

$$X = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{y}_1 \\ \mathbf{x}_2 \\ \mathbf{y}_2 \\ \mathbf{x}_3 \\ \mathbf{y}_3 \end{bmatrix} \tag{1}$$

This is a $m \times n$ matrix where $m = 6$ is the number of measurement types, that is, our three cameras each with two dimensional representation of the data and n is the number of data points taken from the camera over time. This will correspond to the number of frames in the video we consider and will be different for different cases. It makes sense that we gather our data in a matrix because we want to compute SVD to determine the principal components. Also, the different measurements might have a lot of statistical dependency because we are essentially measuring one or two degrees of freedom using three cameras. However, this is exactly why we use PCA to produce a basis other than the horizontal/vertical positions in the three videos in which each direction is independent of the others, that is, a diagonalized covariance matrix. In what follows, we provide a brief overview of the process we undergo in this project.

This project involves two important tasks. The first task is to track the position of the paint can from the video in order to create our data matrix X. The second task is to perform PCA on the data matrix X. A brief overview is provided for each of these tasks.

1.1 Tracking the paint can

The paint can has a flashlight attached to its top. This is to help us track the position of the can in each video frame. We could potentially do this by considering each frame in the video, converting it to grayscale, and setting the position of the can to be the position of the brightest picture in the frame. However, following this procedure blindly immediately lands us into trouble because there would be other bright spots in the video which this procedure picks up on. Hence, we follow a slightly different method to find the position of the paint can. We also sometimes use different techniques for different cameras to track the position. We provide a brief explanation on how we track the can for Test 1.

1.1.1 Test 1

We convert each video frame into an image and grayscale it. We black out certain areas of the image by eye balling the bright regions far away from the paint can. In certain cases, we have a threshold pixel value of 255 and we black out all pixels where it is lower than this value. In other cases, this does not work as the whole frame seems to get blacked out. From here on out, we have two techniques that we use depending on the camera. The first technique is to just find the maximum pixel value in the entire frame and set that to be the position of the can. For the second technique, we create a box of a certain size which we slide across the frame. We take care of considering the corner cases when the box is too wide or too high to fit in the frame. We make sure the box stays within the frame. We find the box with the maximum average pixel value. We turn our focus to this box of the frame only and find the maximum pixel value in the box and set that to be the position of the paint can. This intuitively seems like the correct method because one distinct feature of the paint can in the video is that it has a big region of white color around it. This technique seems to fair well overall but again we have to treat each camera as a separate case and figure out which technique gives a better result.

The techniques we use to track the paint can in other cases are the same. It is possible that we used different areas to block out for different cameras. Also, we tested the two different techniques mentioned above for each camera for the different cases and chose to use the technique that seemed to give the smoothest plot. The code for the four cases for tracking the can are included in Appendix B. Also, the introduction of the parameter size of the box allows us to alter the box and get smoother results by using different window sizes for different window sizes.

1.2 PCA

Once we have tracked the position of the can and collected them into position vectors we stack them to get the matrix in Eq. (1). However, we ensure that the position vectors are aligned, that is, each camera for each dimension starts tracking the position of the can from a set initial position. We choose this initial position to be the position of the highest (negative) displacement in the first 30 frames of the video after eye balling that the can reaches highest displacement in about 30 frames for each of the camera in all the cases. This ensures that lowest and highest points

of the can for each camera roughly matches up. We also ensure that the position vectors are of the same length. Once, we have produced matrix X we calculate the mean of each row and subtract this mean from X. After having done this we perform a reduced SVD on the transpose of the new matrix scaled by a factor of $\frac{1}{\sqrt{n-1}}$.

2 Theoretical Background

2.1 Singular Value Decomposition:

As we learned from our textbook [1], SVD of a $m \times n$ matrix A is the factorization of the form:

$$A = \hat{U}\hat{\Sigma}V^* \tag{2}$$

where $\hat{U}$ is a $m \times n$ matrix with orthonormal columns, $\hat{\Sigma}$ is a $n \times n$ diagonal entries with non negative entries and V is a $n \times n$ unitary matrix. This is called the *reduced SVD* of A and this the form of SVD we use in the project. The following theorem guarantees the existence of such a factorization of any matrix A:

Theorem 1: *Every matrix $A \in \mathbb{C}^{mxn}$ has a SVD. Furthermore, the singular values, that is, diagonal entries of Σ are uniquely determined.*

Another important theorem involving SVD is the following:

Theorem 2: *If the rank of A is r, then there are r nonzero singular values.*

Theorem 2 can help us determine the rank of a system by just considering the singular values. Many other important theorems regarding the SVD are given in [1]. SVD makes it possible for every matrix to be diagonal if the proper bases for the domain and the range are used. SVD provides a type of least-square fitting algorithm, allowing us to project matrix onto low-dimensional representations in a formal, algorithmic way.

2.2 PCA:

We track the position of the paint can and create position vectors which are then gathered into the $m \times n$ matrix X in Eq. (1). This data has two issues: noise and redundancy. If the data is too noisy, then it might be extremely hard to extract the true dynamics. The key measure of the signal-to-noise ratio: SNR $= \sigma^2_{signal}/\sigma^2_{noise}$, where the ratio is given as the ratio of variances of the signal and noise fields. A high SNR gives almost noiseless data whereas a low SNR indicates that the underlying signal is noisy. The second issue is redundancy and the covariance matrix is important in identifying redundancy between data sets.

Covariance measures the statistical dependence/independence between two variables and strongly statistically dependent variables can be considered as redundant observations of the system. In our case, we check the matrix in Eq. (1). The covariance matrix is given by

$$C_X = \frac{1}{n-1}XX^T \tag{3}$$

C_X is a square symmetric matrix whose diagonal represents the variance of particular measurements. The off-diagonal terms are the covariances between measurement types. Large off-diagonal entries correspond to redundancy and large diagonal terms represent the dynamics of interest. Hence, we want to represent C_X so that it is diagonal and the diagonals are ordered from largest to smallest. This is exactly what the SVD does and we use in our project. The SVD diagonalizes by working in the appropriate pair of bases $\hat{U}$ and V and each singular direction captures as much energy as possible as measured by the singular values. If we define the transformed variable

$$Y = \hat{U}^*X \tag{4}$$

where $\hat{U}$ is from Eq.(2). Then:

$$C_Y = \frac{1}{n-1}\hat{\Sigma}^2 \tag{5}$$

The covariance matrix of Y is then diagonal and so we have found basis where each direction is independent of each other. In order to achieve this, we take our matrix X in Eq.(1), standardize it by subtracting the row mean, scale it by a factor of $\frac{1}{n-1}$ then compute the reduced SVD of its transpose to get:

$$X^T = \hat{U}\hat{\Sigma}V^* \tag{6}$$

where each column of $\hat{U}$ contains the paint can's displacement in time along the six PCA modes, the diagonal entries of $\hat{\Sigma}$ contain the singular values whose magnitudes reflect their relative importance. The square of the singular values gives the variance of the data projected onto the PCA modes as shown by Eq.(5).Each column of V contains the directions of the six PCA modes and the products. This means that $V_{1,1}$ and $V_{2,1}$ are the horizontal and vertical components of the first mode in the first video, $V_{3,1}$ and $V_{4,1}$ are the horizontal and vertical components of the first mode in the second video and so on.

3 Algorithm Implementation and Development

We will provide an algorithm for the two different tasks. For tracking the paint can, we only include the technique of using the sliding box. Tracking the paint can using the maximum pixel value position in the whole frame is just a special case when the box is the whole frame.

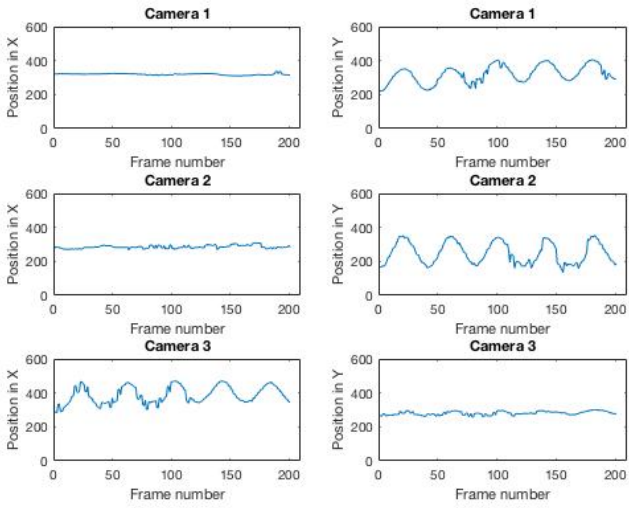
1. Load the video files
2. Extract the height, width and number of frames for each video file
3. Extract the struct data of the images for each frame

3.1 Tracking the paint can

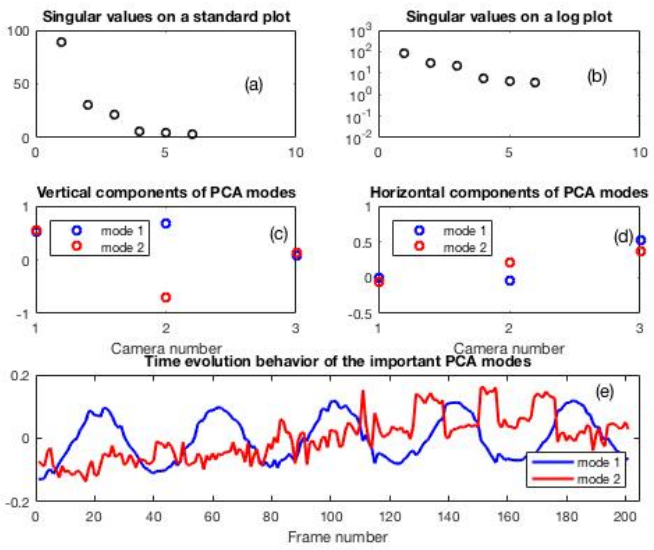
4. Set threshold value and size of the sliding box
5. Initialize position vectors $\mathbf{x}_i$, $\mathbf{y}_i$ for $i = 1, 2, 3$
6. For each frame in camera i, convert to a struct data of an image
7. Convert the image from the previous step to grayscale
8. Use threshold value and/or eye ball to black out bright regions other than the pain can
9. Initialize maximum average to zero and position to an empty vector
10. Create square boxes of the parameter `windowsize`
11. Calculate average of pixel value of the box and update the maximum value if this is greater than the previous maximum value; in this case the position is updated too
12. Slide the box across the frame and make the box fit the frame if the box of window size exceeds the frame
13. Extract the bottom left position of the box and recreate the window
14. Find the maximum pixel value in this new box and retrieve the indices to get the position which are appended to $\mathbf{x}_i$, $\mathbf{y}_i$
15. Gather the position vectors to form X which looks like Eq.(1)

3.2 PCA

16. Find the lowest position of the $\mathbf{x}_i$ position vector and set this to be the initial position for the cameras
17. Truncate the position vectors in such a way that the position vector do not exceed the number of frames for each camera and so that the position vectors have the same length
18. Stack the position vectors to create matrix 1
19. Compute data size and mean for each row and subtract this mean from 1
20. Perform reduced SVD on the transpose of the new matrix scaled by $\frac{1}{\sqrt{n-1}}$ to produce `u`, `s` and `v`
21. Produce the diagonal variances by squaring the diagonal entries of `s`
22. Plot the singular values (entries of `s`), the paint can's displacement in time along important PCA modes (columns of `u`), the direction of the important modes (columns of `v`)



(a) Figure 1 (i)



(b) Figure 1 (ii)

Figure 1: i) Illustration of the motion of the can, ii) singular values on a standard plot (a) and a log plot (b) showing the energy in each POD mode. The bulk of the energy is in the first POD mode. The vertical and horizontal components of the first two modes are illustrated in panel (c) and (d) respectively with their time evolution behavior in panel (e)

4 Computational Results

This section contains the computational results for each of the four cases. We include a plot of the motion of the can for Test 1. The plots for the motion in the cans for other cases are included in Appendix C. We also include a plot of the singular values, the important PCA modes, and the time evolution behavior of the important PCA modes. This includes information from $\hat{U}$, $\hat{\Sigma}$, and V from Eq. (6). Finally, we include a plot of the percentage of variance explained by each mode for Test 1. Such plots for the other cases are included in Appendix C.

4.1 Test 1

This is the case when the paint can is only perturbed in the z-direction. The entire motion should be in the z-direction with simple harmonic oscillation being observation.

Figure 1 (i) illustrates the motion of the can extracted from the camera observations. We can see that for camera 1 and 2 the oscillation seems to be happening in the Y direction and can seems to be more or less fixed in the Y direction. For camera 3, the oscillation occurs in the X direction of the camera and the can seems to be fixed in the Y direction. Figure 1 (ii) illustrates the singular values, the vertical and horizontal components of the first two PCA modes of the three cameras along with their evolution in time. The columns of V from Eq. (6) are the directions of the six PCA modes and we have just plotted the vertical and horizontal directions for the first two PCA modes for the three cameras. The time evolution is the paint can's displacement in along the six PCA modes. a) and b) illustrate that the first PCA mode is significantly more important than the other modes with the value of the first singular value being higher than the rest. Figure 2 shows that more than 82% of the variance is explained by the first mode and we can see that time evolution of the first mode matches up quite well with the motion of the can.

4.2 Test 2

The paint can is perturbed in the same way as Test 1. However, the videos taken by the camera were noisy. This noisy behavior is picked up in the motion of the can in Figure 6 in Appendix C as we see jagged sharp edges. Again, for camera 1 and 2 the motion is the Y direction of the cameras whereas for camera 3 the motion is in the X direction. The noise makes it seem like that there is motion in the other direction too for the cameras.

Figure 3 illustrates the singular values, the vertical and horizontal components of the first two PCA modes of the three cameras along with their evolution in time. The columns of V from Eq. (6) are the directions of the six

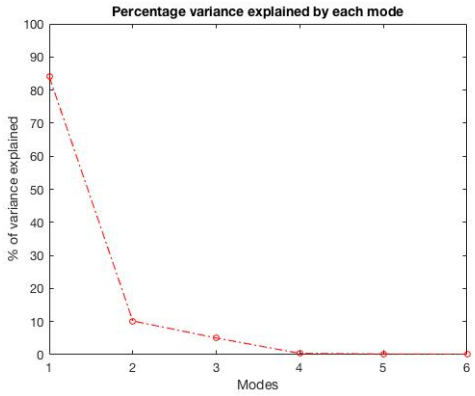


Figure 2: Percentage of variance explained by each mode

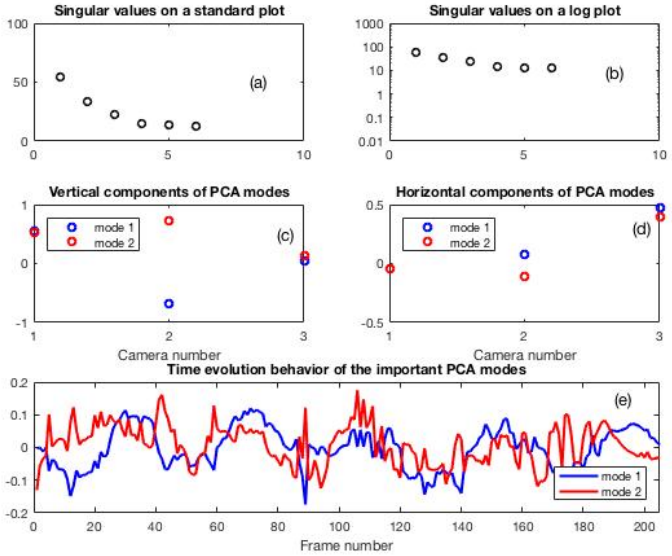


Figure 3: singular values on a standard plot (a) and a log plot (b) showing the energy in each POD mode. The vertical and horizontal components of the first two modes are illustrated in panel (c) and (d) respectively with their time evolution behavior in panel (e)

PCA modes and we have just plotted the vertical and horizontal directions for the first two PCA modes for the three cameras. The time evolution is the paint can's displacement in along the six PCA modes. a) and b) illustrate that the first two PCA modes are significantly more important than the other modes with their values significantly being higher than the rest. Figure 7 in Appendix C shows that about 60% of the variance is explained by the first mode and about 80% of the variance is explained by the first two modes. The evolution of the first two modes seems to match up pretty well with motion of the can.

4.3 Test 3

The paint can is displaced horizontally. There is both a simple harmonic oscillation and a pendulum motion in this case. As seen by Figure 8 in Appendix C, there are oscillations in both X and Y directions for all the cameras.

4.4 Test 4

Finally, in this case we consider when the paint undergoes horizontal displacement along with rotation. Figure 10 in Appendix C shows the motion of the paint can. Again, we oscillatios in both X and Y directions for all cameras.

Figure 5 illustrates the singular values, the vertical and horizontal components of the first two PCA modes of

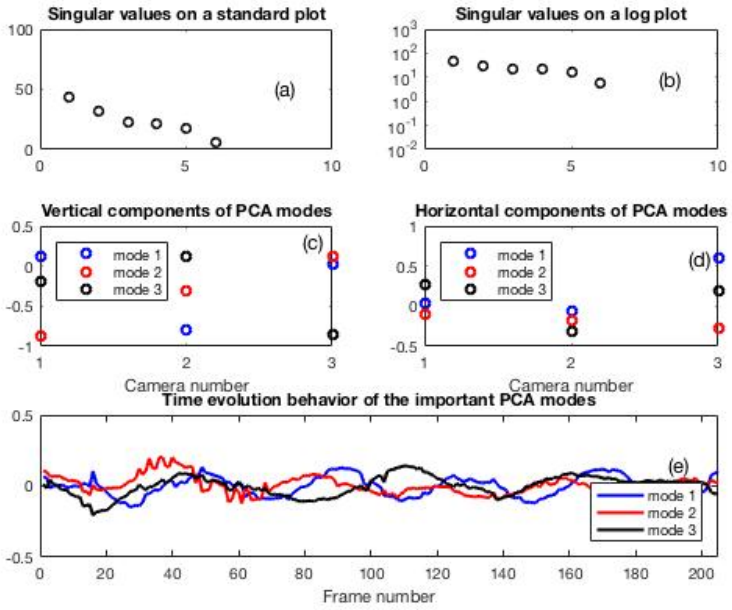


Figure 4: singular values on a standard plot (a) and a log plot (b) showing the energy in each POD mode. The vertical and horizontal components of the first three modes are illustrated in panel (c) and (d) respectively with their time evolution behavior in panel (e)

the three cameras along with their evolution in time. The columns of V from Eq. (6) are the directions of the six PCA modes and we have just plotted the vertical and horizontal directions for the first two PCA modes for the two cameras. The time evolution is the paint can's displacement in along the six PCA modes. a) and b) illustrate that the first two PCA modes are significantly more important than the other modes with their values higher than the rest. Figure 11 in Appendix C shows that about 80% of the variance is explained by the first two modes. The evolution of the first two modes seems to match up pretty well with motion of the can.

5 Summary and Conclusions

This project demonstrates how powerful PCA is in extracting the underlying dynamics of a system even when the governing principles are not known. PCA is able to do this even with noisy data. We see that one or two modes seems to be enough in the most cases to view the motion of the paint can. However, there are some issues that need to be addressed. Some dynamics could have been extracted with fewer modes (Test 3) but this may have been due to the inaccuracy in tracking the can. The tracking often moves from the flashlight to the white part of the paint can which gives the jagged edges in Figure 6. However, the positions seems to be of the can and not some other part of the frame and so we stick with it. This technique could be made better for smoother plots with time and more sophisticated algorithms.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data.* Oxford University Press, 2013.

Appendix A MATLAB Functions

This section contains important MATLAB functions with a brief implementation explanation.

- `[X,map] = frame2im(F)` returns image data associated with a single frame `F` in a video. `X` is the indexed image data and `map` is the associated `colormap` from the single frame `F`. In our case, `colormap` is an empty field because the image in single frame is a truecolor (RGB) image.

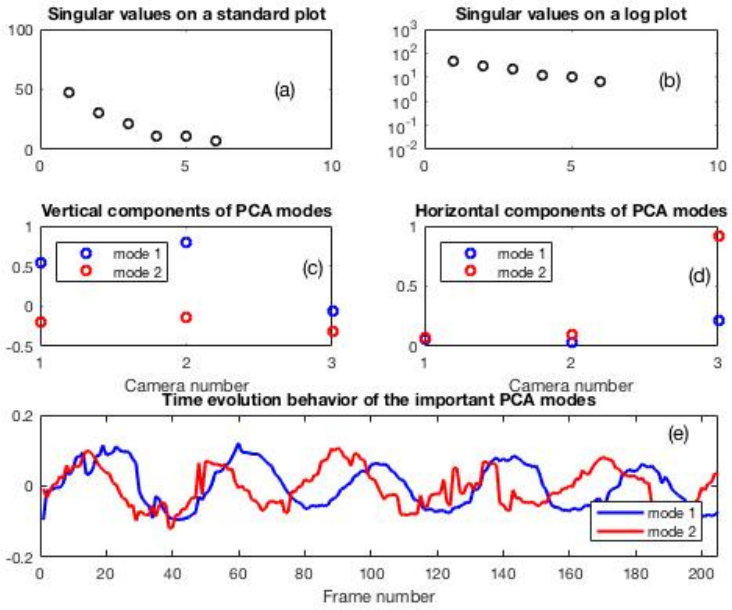


Figure 5: singular values on a standard plot (a) and a log plot (b) showing the energy in each POD mode. The vertical and horizontal components of the first two modes are illustrated in panel (c) and (d) respectively with their time evolution behavior in panel (e)

- `I = rgb2gray(X)` converts the the truecolor image `X` to the grayscale image `I`.
- `M = mean(X,dim)` returns the mean along dimension `dim`. In our case, `X` is a matrix and we set `dim` = 2. Then, `M = mean(X,2)` returns a column vector containing the mean of each row.
- `B = repmat(A,r1, r2, ..., rn)` returns an array containing copies of `A` and the scalars `r1, r2,..., r2` describe how copies of `A` are arranged in each dimension.
- `[U,S,V] = svd(X)` performs reduced SVD on the matrix `X` and returns `U`, `S` and `V` which are $\hat{U}$, $\hat{\Sigma}$ and V from Eq.(2).
- `x = diag(A)` returns a column vector `x` of the main diagonal elements of `A`

Appendix B MATLAB Code

We provide the code used in the project. We provide the full code for Test 1. The codes for the other cases are almost exactly the same with the only difference being in the tracking of the paint can. For the Test 2, we only provide code for the tracking of the paint can. This is to demonstrate how different techniques were used and how different areas were blacked out to extract the motion of the paint can.

```
%Shabab Ahmed
%AMATH 582
%HW 3 (PCA)
%-------------------------------------------------------------------------

% Ideal case

% Loading the data

clear all; close all; clc

for k = 1:3
```

```
    load(['cam' num2str(k) '_1'])
end

% Changing variable name

vidFrames1 = vidFrames1_1;
vidFrames2 = vidFrames2_1;
vidFrames3 = vidFrames3_1;


%[height, width, [R,G,B], number of frames] = size(vidFrames)
[h1, w1, c1,n1] = size(vidFrames1);
[h2, w2, c2,n2] = size(vidFrames2);
[h3, w3, c3,n3] = size(vidFrames3);


% obtain number of frames for the cameras
numFrames1 = n1;
numFrames2 = n2;
numFrames3 = n3;


for k = 1 : numFrames1
    mov1(k).cdata = vidFrames1(:,:,:,k);
    mov1(k).colormap = [];
end

for k = 1 : numFrames2
    mov2(k).cdata = vidFrames2(:,:,:,k);
    mov2(k).colormap = [];
end

for k = 1 : numFrames3
    mov3(k).cdata = vidFrames3(:,:,:,k);
    mov3(k).colormap = [];
end

% threshold pixel value
threshold = 255;

% Initialize the position vectors for cam1

X1 = [];
Y1 = [];

for k=1:numFrames1
    gscale = frame2im(mov1(k));
    gscale = rgb2gray(gscale);  % convert to grayscale
    gscale(gscale<threshold) = 0; % zero (black) out pixel below threshold
    gscale(:,1:275) = 0;        % zero(black) out specific region in video
    [m, ind] = max(gscale(:)); % find maximum pixel value and index
    [x1, y1] = ind2sub(size(gscale), ind);
    X1 = [X1, x1];
    Y1 = [Y1, y1];
end
```

```
%-----------------------------------------------------------------------
% Initialize position vectors for camera 2

X2 = [];
Y2 = [];

% windowsize for sliding window across the frame
windowsize = 10;

for k=1:numFrames2
    gscale = frame2im(mov2(k));
    gscale = rgb2gray(gscale);  % convert to grayscale
    maxavg = 0;                 % set maximum average pixel value of window
    pos = [];                   % initial position
    for j = 1:windowsize:h1
        for i = 1:windowsize:w1

            % if height and width out of bounds set appropriate window
            if (j+windowsize>h1) & (i+windowsize>w1)
                box = gscale(j:h1,i:w1);

            % if height out of bounds set appropriate window
            elseif (j+windowsize>h1)
                box = gscale(j:h1,i:i+windowsize);

            % if width out of bounds set appropriate window
            elseif (i + windowsize > w1)
                box = gscale(j:j+windowsize, i:w1);

            % normal window
            else
                box = gscale(j:j+windowsize, i:i+windowsize);
            end

            % average the pixel value in the window
            avg = sum(box(:));
            avg = avg/(windowsize*windowsize);

            % update maximum average and position found for each window

            if avg > maxavg
                maxavg = avg;
                pos = [j,i];


            end
        end
    end

    % extract bottom left position of box with maximum average pixel value
    j= pos(1);
    i = pos(2);

    %recreate the window of maximum average pixel value
    subframe = gscale(j:j+windowsize,i:i+windowsize);
```

```
    % find maximum pixel intensity in the window
    [m, ind] = max(subframe(:));

    % find corresponding indices
    [x2 y2] = ind2sub(size(subframe), ind);
    X2 = [X2, x2+j];
    Y2 = [Y2, y2+i];
end


%-------------------------------------------------------------------------
% Initializing position vectors for camera 3

X3 = [];
Y3 = [];


% windowsize for sliding window across the frame
windowsize = 5;

for k=1:numFrames3
    gscale = frame2im(mov3(k));
    gscale = rgb2gray(gscale);  % convert to grayscale
    % zero (black) out bright spots for each frame
    gscale(1:175,:) = 0;
    gscale(:,1:230) = 0;
    maxavg = 0;                 % set maximum average pixel value of window
    pos = [];                   % initial position
    for j = 1:windowsize:h1
        for i = 1:windowsize:w1

            % if height and width out of bounds set appropriate window
            if (j+windowsize>h1) & (i+windowsize>w1)
                box = gscale(j:h1,i:w1);

            % if height out of bounds set appropriate window
            elseif (j+windowsize>h1)
                box = gscale(j:h1,i:i+windowsize);

            % if width out of bounds set appropriate window
            elseif (i + windowsize > w1)
                box = gscale(j:j+windowsize, i:w1);

            % normal window
            else
                box = gscale(j:j+windowsize, i:i+windowsize);
            end

            % average the pixel value in the window
            avg = sum(box(:));
            avg = avg/(windowsize*windowsize);

            % update maximum average and position found for each window

            if avg > maxavg
                maxavg = avg;
```

```
                pos = [j,i];


            end
        end
    end

    % extract bottom left position of box with maximum average pixel value
    j= pos(1);
    i = pos(2);

    %recreate the window of maximum average pixel value
    subframe = gscale(j:j+windowsize,i:i+windowsize);

    % find maximum pixel intensity in the window of maximum average
    [m, ind] = max(subframe(:));

    % find corresponding indices
    [x3 y3] = ind2sub(size(subframe), ind);
    X3 = [X3, x3+j];
    Y3 = [Y3, y3+i];
end



% Aligning the position vectors so that the lowest point matches up

% find the lowest position in the first 30 frames

[m1, i1] = min(X1(1:30));
[m2, i2] = min(X2(1:30));
[m3, i3] = min(X3(1:30));


% align and crop the position vectors to 200 frames

X1 = X1(i1:i1+200); Y1 = Y1(i1:i1+200);
X2 = X2(i2:i2+200); Y2 = Y2(i2:i2+200);
X3 = X3(i3:i3+200); Y3 = Y3(i3:i3+200);


% Plotting the position vectors

figure(2)
% Camera 1
subplot(3,2,1)
plot(Y1)
axis([0 210 0 600])
ylabel('Position in X')
xlabel('Frame number')
title('Camera 1')

subplot(3,2,2)
plot(X1)
axis([0 210 0 600])
ylabel('Position in Y')
xlabel('Frame number')
```

```
title('Camera 1')

% Camera 2
subplot(3,2,3)
plot(Y2)
axis([0 210 0 600])
ylabel('Position in X')
xlabel('Frame number')
title('Camera 2')

subplot(3,2,4)
plot(X2)
axis([0 210 0 600])
ylabel('Position in Y')
xlabel('Frame number')
title('Camera 2')

% Camera 3
subplot(3,2,5)
plot(Y3)
axis([0 210 0 600])
ylabel('Position in X')
xlabel('Frame number')
title('Camera 3')

subplot(3,2,6)
plot(X3)
axis([0 210 0 600])
ylabel('Position in Y')
xlabel('Frame number')
title('Camera 3')



X = [X1;Y1;X2;Y2;X3;Y3]; % creating the data matrix
[m,n] = size(X);  % compute data size
mn = mean(X,2); % mean for each row
X = X - repmat(mn,1,n); % subtract mean

[u,s,v] = svd(X'/sqrt(n-1),0); % perform the svd
lambda = diag(s).^2;  % produce diagonal variances

sig =diag(s); % singular values



% Plot percentage variance explained by each mode
figure(3)
plot(lambda/sum(lambda)*100, '-.or')
axis([1 6 0 100])
set(gca,'Fontsize',[13],'Ytick',[0 10 20 30 40 50 60 70 80 90 100],...
'Xtick',[1 2 3 4 5 6]);
ylabel('% of variance explained')
xlabel('Modes')
title('Percentage variance explained by each mode')
```

```
% Plot singular values on a standard and log plot
figure(4)

% Standard plot
subplot(3,2,1)
plot(sig, 'ko','Linewidth',[1.5])
axis([0 10 0 100])
set(gca,'Fontsize',[10],'Xtick',[0 5 10])
text(8,50,'(a)','Fontsize',[13])
title('Singular values on a standard plot')

% Log plot
subplot(3,2,2), semilogy(sig,'ko','Linewidth',[1.5])
axis([0 10 10^(-2) 10^3])
set(gca,'Fontsize',[10],'Ytick',[10^(-2) 10^(-1) 10^(0) 10^(1) 10^(2)...
10^3],'Xtick',[0 5 10]);
text(8,8,'(b)','Fontsize',[13])
title('Singular values on a log plot')

% Plot vertical components of first two PCA modes
subplot(3,2,3)
plot([v(1,1),v(3,1), v(5,1)], 'bo', 'Linewidth', [2]); hold on;
plot([v(1,2), v(3,2),v(5,2)],'ro', 'Linewidth',[2])
legend('mode 1', 'mode 2','Location','NorthWest')
set(gca,'Fontsize',[10], 'Xtick', [1 2 3])
text(2.8,0.5,'(c)','Fontsize',[13])
xlabel('Camera number')
title('Vertical components of PCA modes')

% Plot horizontal components of first two PCA modes
subplot(3,2,4)
plot([v(2,1),v(4,1), v(6,1)], 'bo','Linewidth',[2]); hold on;
plot([v(2,2), v(4,2),v(6,2)],'ro', 'Linewidth',[2])
set(gca,'Fontsize',[10], 'Xtick', [1 2 3])
legend('mode 1', 'mode 2','Location','NorthWest')
text(2.8,0.6,'(d)','Fontsize',[13])
title('Horizontal components of PCA modes')
xlabel('Camera number')

% Plot of time evolution behavior of PCA modes

subplot(3,1,3)
plot(u(:,1), 'b', 'Linewidth',[2]); hold on
plot(u(:,2), 'r', 'Linewidth',[2]);
xlim([0 205])
text(190,0.15,'(e)','Fontsize',[13])
legend('mode 1', 'mode 2','Location','SouthEast')
title('Time evolution behavior of the important PCA modes')
xlabel('Frame number')


% Test 2(Noisy case)

clear all; close all; clc

for k = 1:3
    load(['cam' num2str(k) '_2'])
```

```
end
vidFrames1 = vidFrames1_2;
vidFrames2 = vidFrames2_2;
vidFrames3 = vidFrames3_2;

%[height, width, [R,G,B], number of frames]

[h1, w1, c1,n1] = size(vidFrames1);
[h2, w2, c2,n2] = size(vidFrames2);
[h3, w3, c3,n3] = size(vidFrames3);


% obtain number of frames for the cameras
numFrames1 = n1;
numFrames2 = n2;
numFrames3 = n3;

numFrames = min([numFrames1, numFrames2, numFrames3]);

for k = 1 : numFrames1
    mov1(k).cdata = vidFrames1(:,:,:,k);
    mov1(k).colormap = [];
end

for k = 1 : numFrames2
    mov2(k).cdata = vidFrames2(:,:,:,k);
    mov2(k).colormap = [];
end

for k = 1 : numFrames3
    mov3(k).cdata = vidFrames3(:,:,:,k);
    mov3(k).colormap = [];
end


% Initialize position vectors for camera 1

X1 = [];
Y1 = [];

% window (box) size
windowsize = 5;

for k=1:numFrames1
    gscale = frame2im(mov1(k));
    gscale = rgb2gray(gscale);  % convert to grayscale
    gscale(:,1:275) = 0;        % zero(black) out specific region in video
    maxavg = 0;                 % set maximum average pixel value of window
    pos = [];                    % initial position


    for j = 1:windowsize:h1
        for i = 1:windowsize:w1
            % if height and width out of bounds set appropriate window
            if (j+windowsize>h1) & (i+windowsize>w1)
                box = gscale(j:h1,i:w1);
```

```
            % if height out of bounds set appropriate window
            elseif (j+windowsize>h1)
                box = gscale(j:h1,i:i+windowsize);

            % if width out of bounds set appropriate window
            elseif (i + windowsize > w1)
                box = gscale(j:j+windowsize, i:w1);

            % normal window
            else
                box = gscale(j:j+windowsize, i:i+windowsize);
            end
            % average the pixel value in the window
            avg = sum(box(:));
            avg = avg/(windowsize*windowsize);

            % update maximum average and position found for each window
            if avg > maxavg
                maxavg = avg;
                pos = [j,i];


            end
        end
    end

    % extract bottom left position of box with maximum average pixel value
    j= pos(1);
    i = pos(2);

    %recreate the window of maximum average pixel value
    subframe = gscale(j:j+windowsize,i:i+windowsize);

    [m, ind] = max(subframe(:)); % find maximum pixel intensity
    [x1 y1] = ind2sub(size(subframe), ind); % find corresponding indices
    X1 = [X1, x1+j];
    Y1 = [Y1, y1+i];
end

%-------------------------------------------------------------------------
% Initialize position vectors for camera 2

X2 = [];
Y2 = [];

% window (box) size

windowsize = 5;

for k=1:numFrames2
    gscale = frame2im(mov2(k));
    gscale = rgb2gray(gscale);  % convert to grayscale

    % Zero (black) out bright pixels that are not the paint can

    gscale(:,1:275) = 0;
    gscale(:,330:end) =0;
```

```
    gscale(1:200,:) = 0;

    maxavg = 0;
    pos = [];

    for j = 1:windowsize:h1
        for i = 1:windowsize:w1
            if (j+windowsize>h1) & (i+windowsize>w1)
                box = gscale(j:h1,i:w1);
            elseif (j+windowsize>h1)
                box = gscale(j:h1,i:i+windowsize);
            elseif (i + windowsize > w1)
                box = gscale(j:j+windowsize, i:w1);
            else
                box = gscale(j:j+windowsize, i:i+windowsize);
            end

            avg = sum(box(:));
            avg = avg/(windowsize*windowsize);
            if avg > maxavg
                maxavg = avg;
                pos = [j,i];


            end
        end
    end
    j= pos(1);
    i = pos(2);
    subframe = gscale(j:j+windowsize,i:i+windowsize);
    [m, ind] = max(subframe(:)); % find maximum pixel intensity
    [x2 y2] = ind2sub(size(subframe), ind); % find corresponding indices
    X2 = [X2, x2+j];
    Y2 = [Y2, y2+i];
end

%---------------------------------------------------------------------
% Initialize position vectors for camera 3

X3 = [];
Y3 = [];

% window (box) size

windowsize = 5;

for k=1:numFrames3
    gscale = frame2im(mov3(k));
    gscale = rgb2gray(gscale);  % convert to grayscale

    % Zero (black) out bright pixels that are not the paint can

    gscale(1:150,:) = 0;
    gscale(:,1:230) = 0;
    maxavg = 0;
    pos = [];
    for j = 1:windowsize:h1
```

```
        for i = 1:windowsize:w1
            if (j+windowsize>h1) & (i+windowsize>w1)
                box = gscale(j:h1,i:w1);
            elseif (j+windowsize>h1)
                box = gscale(j:h1,i:i+windowsize);
            elseif (i + windowsize > w1)
                box = gscale(j:j+windowsize, i:w1);
            else
                box = gscale(j:j+windowsize, i:i+windowsize);
            end

            avg = sum(box(:));
            avg = avg/(windowsize*windowsize);
            if avg > maxavg
                maxavg = avg;
                pos = [j,i];


            end
        end
    end
    j= pos(1);
    i = pos(2);
    subframe = gscale(j:j+windowsize,i:i+windowsize);
    [m, ind] = max(subframe(:)); % find maximum pixel intensity
    [x3 y3] = ind2sub(size(subframe), ind); % find corresponding indices
    X3 = [X3, x3+j];
    Y3 = [Y3, y3+i];
end
```

Appendix C Plots

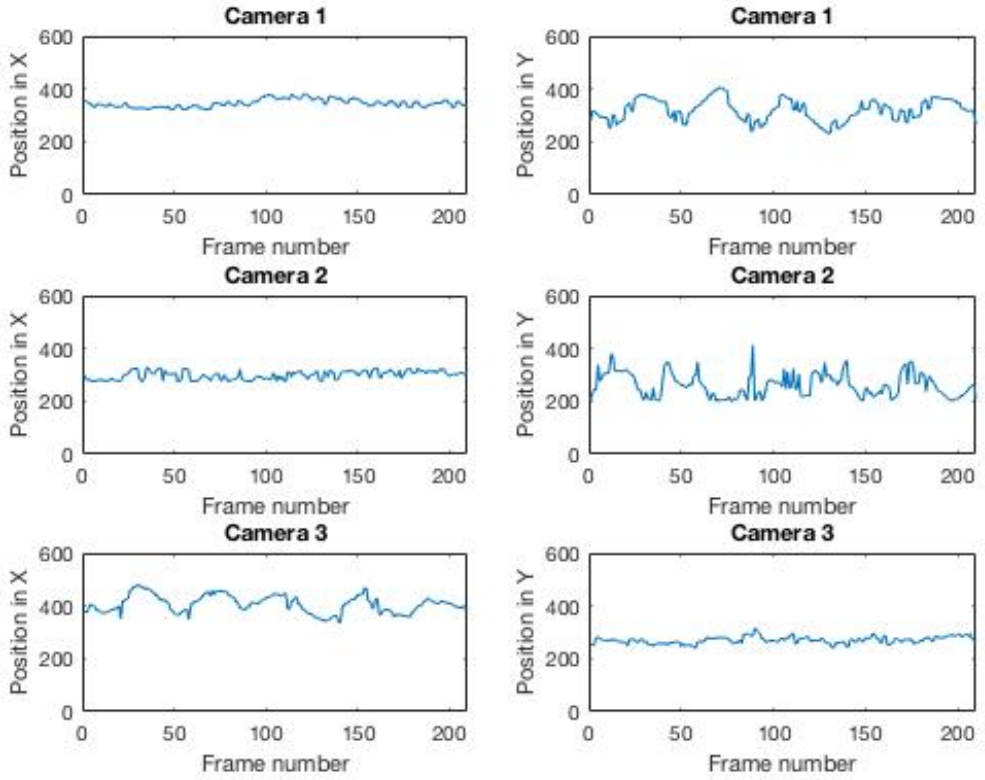


Figure 6: Motion of the paint can (Test 2)

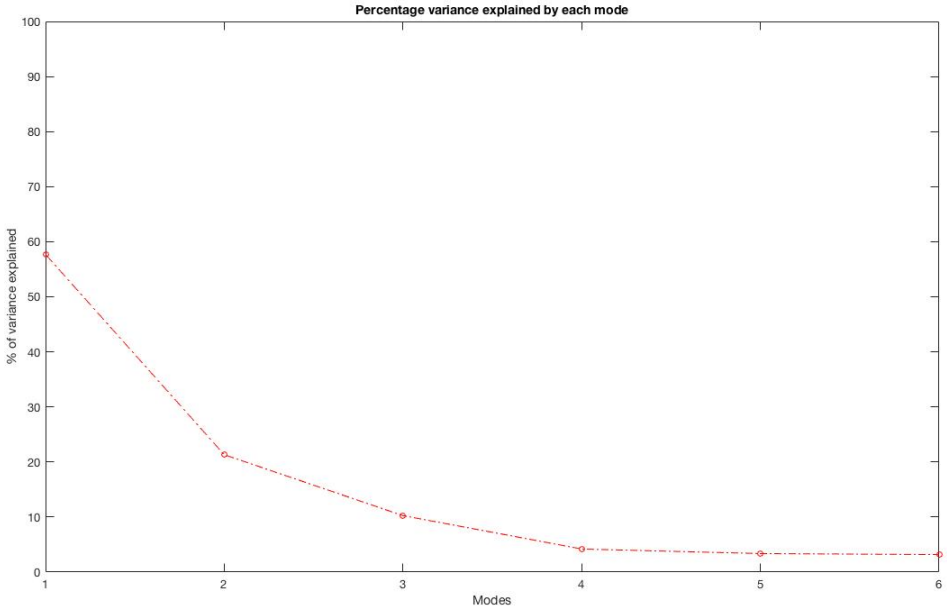


Figure 7: Percentage of variance explained by each mode (Test 2)

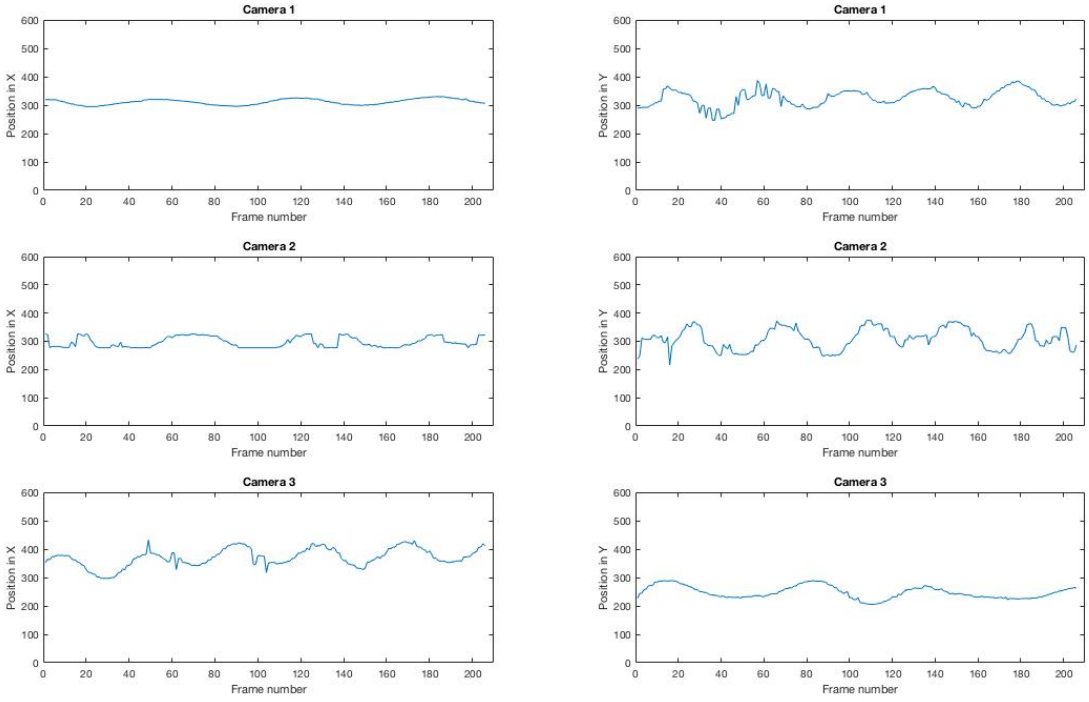


Figure 8: Motion of the paint can (Test 3)

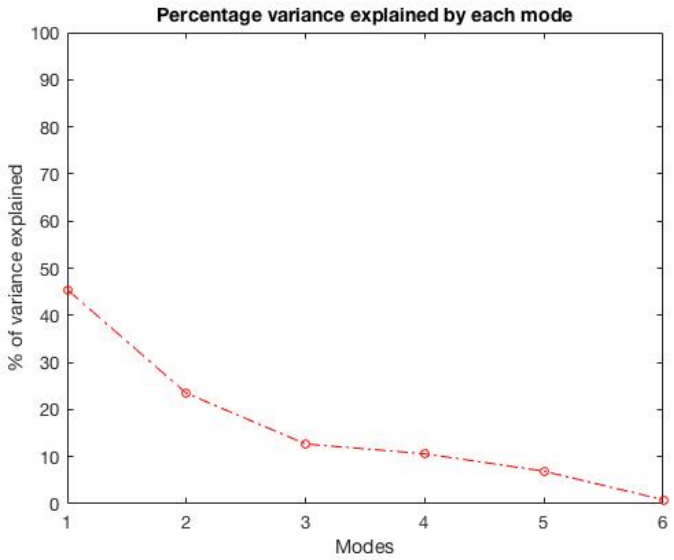


Figure 9: Percentage of variance explained by each mode (Test 3)

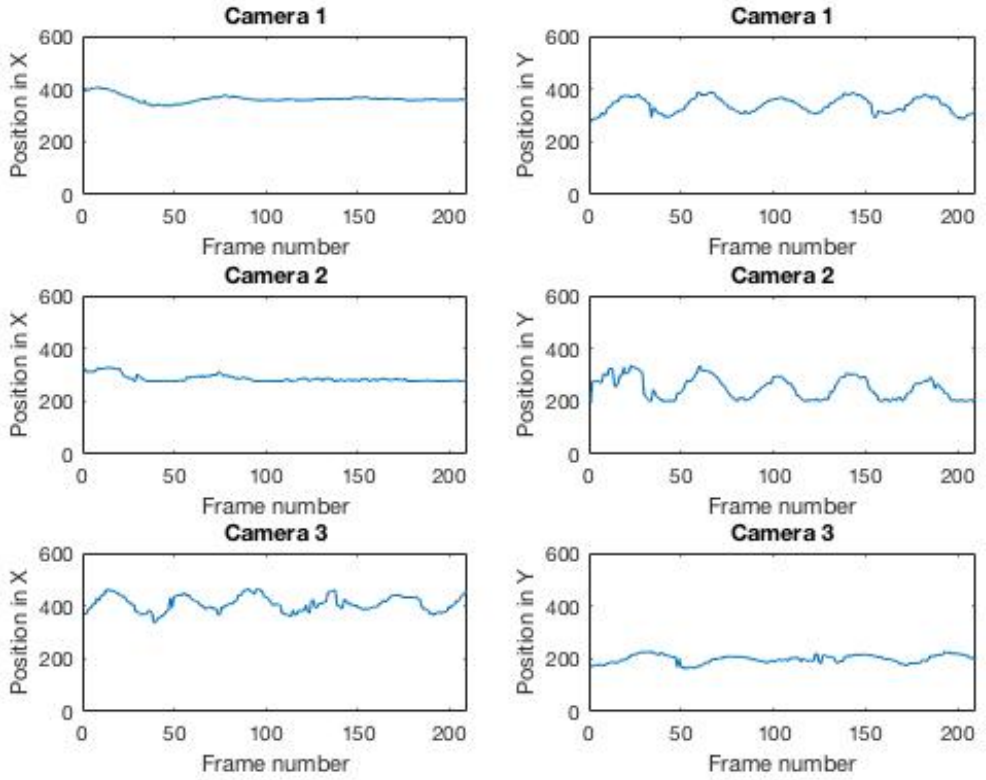


Figure 10: Motion of the paint can (Test 4)

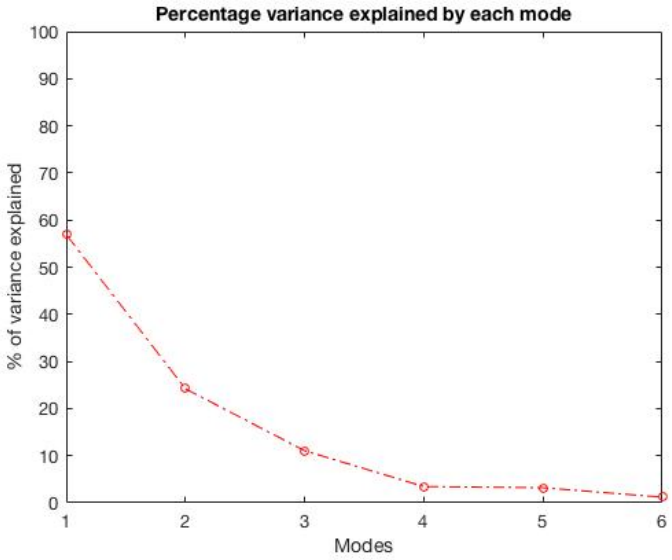


Figure 11: Percentage of variance explained by each mode (Test 4)