Identifying Dynamics Using Principle Component Analysis

Elliot Jennis

February 2021

Abstract

Principle Component Analysis (PCA) was used to determine the underlying dynamics of a mass moving within several different video frames. The motion of this mass varied over 4 separate experiments attempting to capture harmonic motion, pendulum motion, and rotational motion of the mass. The motion was captured using three cameras at three separate angles to ensure complete capture of the motion. Algorithmic tools were used to extract location data from each video were developed. PCA proved to be a robust algorithmic tool at extracting the dynamics of a system when little noise is present. Noisy data can still contain useful information, with harmonic structure still being present, however the drastic differences in clarity between the low noise and high noise data sets shows the sensitivity of PCA to noise.

Introduction and Overview

Principle component analysis (PCA) is a powerful tool used in data science. It is an algorithmic process that uses linear algebra theorems to extract useful information from multidimensional data sets. These data sets can be measurements from physical sensors, image files, statistical data on populations, or other strings of data representing different experiments or collected information. To demonstrate how to use the PCA to uncover the dynamics of an experiment, an example problem with known dynamics is presented. Simple harmonic motion is a very well understood and documented physical phenomena with clear solutions and known equations of motion. PCA conducted on imaging data of a body in simple harmonic motion should be able to reveal periodic oscillations with a single degree of freedom. Further analysis on the addition of noise into the data, as well as adding degrees of freedom through the introduction of rotation and pendulum motion are also considered.

Theoretical Background

To understand the PCA algorithm, first a matrix transformation formula must be introduced. This transformation is called the Singular Value Decomposition or SVD. In a general context, a matrix transform is tool used to rotate and/or stretch an existing vector.

$$\mathbf{A} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} \alpha & 0 \\ 0 & \alpha \end{bmatrix} \tag{1}$$

Equation line 1 shows two different types of transformation matrices. **A** represents a rotation matrix about angle θ. **B** represents a stretching matrix by the coefficient α. On a fundamental level, these two processes (rotation and stretching) are the purpose of the SVD.

A full description of the SVD will begin with its definition. As shown in equation 2, the SVD is a factorization of the matrix **A** into three sub-components $\boldsymbol{V^*}$, $\boldsymbol{\Sigma}$, and **U** where **V** $\in \mathbb{C}^{nxn}$ is a unitary matrix, $\boldsymbol{\Sigma} \in \mathbb{C}^{mxn}$ is a diagonal matrix, and $\mathbf{U} \in \mathbb{C}^{mxm}$ is also unitary[1].

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

In practical use, matrix **A** represents a data matrix where each column of **A** represents a given measurement reshaped into a $1xn$ column vector. There are a total of m columns representing an m-dimensional data set. An example would be single frame of a movie being reshaped into a single column vector in **A**. The resolution of the frame being $480x640$ would represent a $307200x1$ column vector. The rest of the frames in the movie would fill out **A**. Matrix **U**, also known as the left singular vector matrix, represents a basis for the dimensional space **A** is in[1]. This change of basis rotates the data within the **A** by changing its coordinate frame. The $\boldsymbol{\Sigma}$ matrix represent singular values for **A** on its diagonal. These values are ordered $\sigma_1 > \sigma_2 > ...\sigma_n$ where σ_k are all non negative values. These values are the singular values for the SVD. In a broad sense, they indicate the relative importance of the columns of **U** and **V** to the reconstruction of **A**. The first entry σ_1 being the most important and subsequent entries being less and less important.

Another way to think of the SVD factorization is that it provides a set of n rank 1 matrices that add up to **A**. This set is of the form $\boldsymbol{A} = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + ... + \sigma_n u_n v_n^T$. Each subsequent term in the set increases the accuracy of the sets approximation of **A**. A fully accurate depiction of **A** contains all n terms but approximations for **A** can be made using the first few terms of the set.

With this understanding of the SVD, PCA can be more fully understood. The goal of PCA is to remove redundancy and identify the signals with the most variance within a given data matrix [1]. An easy way to identify redundancy within the data matrix to to find the covariance between data sets. The variance within a data set is defined in equation 2[1] while the covariance defined in equation 3[1] where a and b represent two columnar data sets. Equation 4 represents the generalized covariance matrix for a given data matrix **X**.

$$\sigma_a^2 = \frac{1}{n-1} aa^T \tag{3}$$

$$\sigma_a^2 b = \frac{1}{n-1} ab^T \tag{4}$$

$$\boldsymbol{C_X} = \frac{1}{n-1} \boldsymbol{XX^T} \tag{5}$$

$\boldsymbol{C_x}$ is a square symmetric mxm matrix whose off diagonal terms represent the covariances between different combinations of measurements and the diagonal terms represent the variances of individual measurements. Large values in the off diagonal terms are typically associated with redundant data while small values in these locations indicate likely statistical independence. For the diagonal terms, large values which represent large variances, help identify the dynamics of interest within a given data set[1]. This can help in the specific case of this paper, identifying the degrees of freedom of a moving mass from video data.

Algorithm Implementation and Development

There are multiple algorithms in use within the proposed solution to this problem that require further explanation. Additionally, there were four separate cases to consider. Thankfully the structure of the approach for each case was largely the same.

1. **Convert the raw video files into matrices for manipulation and analysis**
2. **Track the mass in x and y space as it moves through each frame**
3. **Conduct PCA on x and y location data**

Surprisingly the most challenging part of this task wasn't the PCA analysis which is the theoretical focus of this paper, it was the location tracking of the mass. Luckily, there were reference points consistent thought the frames that allowed for accurate location tracking. Specifically, the approach detailed in this paper took advantage of the light attached to the mass to achieve accurate location tracking. The first task was to convert the video file into a matrix with dimensions equivalent to the resolution of the video file. The process for accomplishing this is detailed in Listing 1.

Once each video had been converted to a three dimensional matrix (480x640xNumFrames), the tracking algorithm could be implemented. Every value in the matrix represented a pixel location. The values ranged from 0-255 with 0 being black and 255 being white. The light atop the mass was typically the brightest object in frame thus creating a cluster of numbers close to 255 that was easily identifiable.

However, it was not enough to simply search for the brightest object in each frame and call that done. To reduce the likelihood of miss-identification of the background as the light, the following algorithm was implemented (Listing 2).

1. **Locate the light in x and y space for the first frame of each video by searching for the brightest object in a cropped matrix.** The first frame was cropped to only include the pixels containing the mass and the light ensuring the light location was accurate. Then a correction factor was added to the averaged light location to match the index of the light in the cropped frame to the index of the light in the full frame.

2. **Use a bounding box to find the location of the light in each subsequent frame.** The same process used to detect the white pixels in frame one is used to detect the location of white pixels in the subsequent frames. However, the frames following frame one were all uncropped allowing for a full frame search of the white pixels. The white pixel locations are temporarily stored in two vectors, one representing the x index and one the y index of the white pixels. A bounding box of size nxn (30x30

in the listing) centered at location of the average pixel value from the previous frame was also created using a series of logic statements. This bounding box defines which values in the x and y location vectors are kept, with the rest of the values being deleted from the vectors. The remaining values in the location vectors are then averaged. This average location becomes the center of the bounding box for the next frame and so on.

3. **For frames where no values are detected, interpolation is used to fill in the gaps.**

The final algorithm is the PCA algorithm. A data matrix $\mathbf{X}$ was constructed with six rows corresponding to the x and y location data for all three camera angles. The mean value of this matrix was extracted and then deleted from each entry within it to recenter the data cloud at zero mean. The SVD of the data matrix was computed using the 'econ' command to reduce the computational time of the algorithm. Paying close attention to the $\mathbf{\Sigma}$ matrix, the modal energies of the principle components were plotted on a log graph by dividing $\mathbf{\Sigma}$ by the sum of its components. The dominant modes for each of the 4 problems were then identified. These dominant modes were then projected back onto the data matrix and the accuracy of the modal depiction of the data was directly compared to the original. This helped determine degrees of freedom for each of the four problems.

Computational Results

Problem 1

The dynamics present in the first problem should indicate simple harmonic motion in a single degree of freedom in the ideal case. The mass is only moving up and down, there is very little noise present in the data, and camera motion is kept to a minimum. Once the data was processed and decomposed using the SVD, the modal energies present within the data were plotted on a log graph. Surprisingly, there were two modes that captured over 90 percent of the data instead of just one (Figure 1). This indicates that there is some kind of additional motion present in the mass. The projection of the dominant modes onto the original data is shown in Figure 2. Most interestingly, this second degree of freedom is out of phase with the first by about 90 degrees. This seems to indicate that this second mode is the derivative of the first as a sin wave would be 90 degrees out of phase with a cos wave.

Problem 2

The second batch of data was taken with noise inherent in the system due to excessive camera motion. This noise expectantly caused challenges when attempting to conduct PCA. The modal energies were plotted again (Figure 3). This time the energies followed a much more coherent curve with no obvious drop off between dominant and non dominant modes. Most of the energy is still concentrated in the first two, but the distribution is much more uniform. When these modes were projected onto the data (Figure 4) the inherent noise is much more apparent. The underlying periodic structure is still observable, but the noise has drastically reduced its clarity.

Problem 3

The addition of pendulum motion for this case, would indicate that additional dynamics will be present. A modal energy plot was produced revealing that this was indeed the case with most of the dynamics occurring in the first three modes (Figure 5). The projection of the modes once again revealed the same two periodic oscillations 90 degrees out of phase with each other shown in blue and yellow. The third dynamic is the line in orange representing the pendulum motion in the system. This is clear due to its longer and variable periods.

Problem 4

The final piece of the puzzle, rotation, was introduced in this problem. This was a much more difficult oscillatory behavior to track due to the small motions and the highly accurate and precise tracking it required. The degree of confidence in these results is lower than those of the other two problems. The modal energies plot is different than the rest of the plots created so far (Figure 7). The plot is nearly linear with no obvious gap between modes. The assumption made is that the first four are the dominant modes with the understanding that rotation represents one additional degree of freedom from the three dominant modes in the previous problem. The projection of the modes onto the data has some familiar structures. The blue and orange curves continue to represent the simple harmonic oscillations from problem one. The yellow curve seems to show the pendulum motion due to its periodic nature as well as its central location about the origin. The final curve shown in purple is an attempt at describing the dynamics of rotation. There is periodic behavior to this curve, however it seems to drift away from this behavior as time progresses. This might be due to poor rotation tracking for this problem, or the slowing down of rotation over the duration of the data acquisition period.

Summary and Conclusions

The PCA algorithms used in this paper helped reveal the dynamics of each of the four problems presented in this paper. Simple harmonic motion has a very clear consistent signature throughout the four problems. Even when corrupted with noise in the data, the periodic nature of the motion is clear. The addition of further degrees of freedom can still be captured and characterized by PCA as long as the method used to extract the data is precise enough. This is likely what caused the unclear results in problem 4, the algorithm used to extract the location of the mass in each frame was not sensitive enough to accurately extract rotation data.

References

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

A Figures

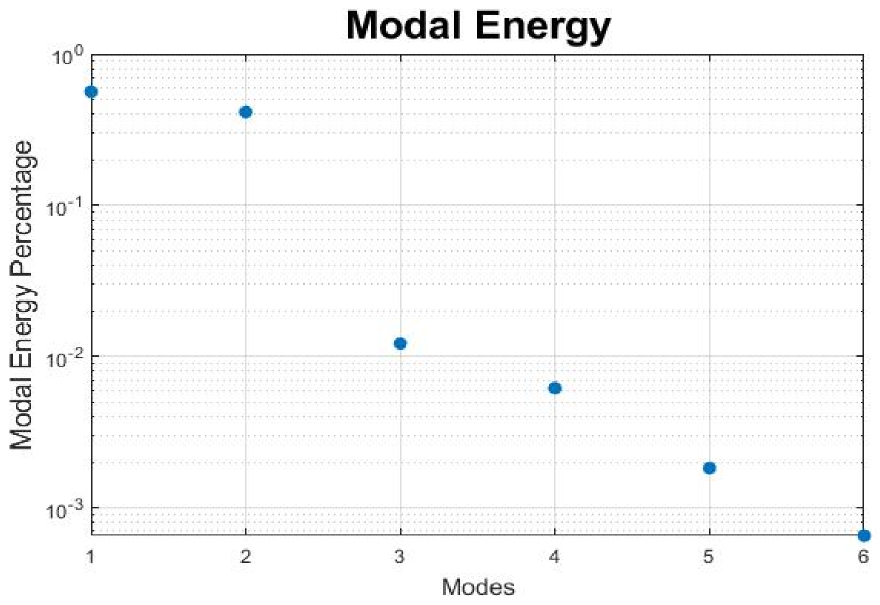

Figure 1: Modal Energies of problem 1

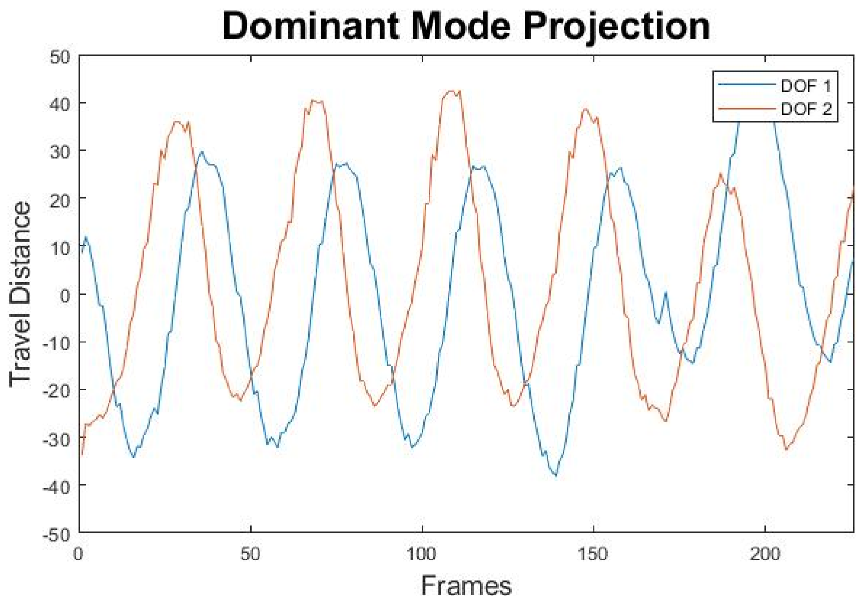

Figure 2: Modal projection of location data for problem 1

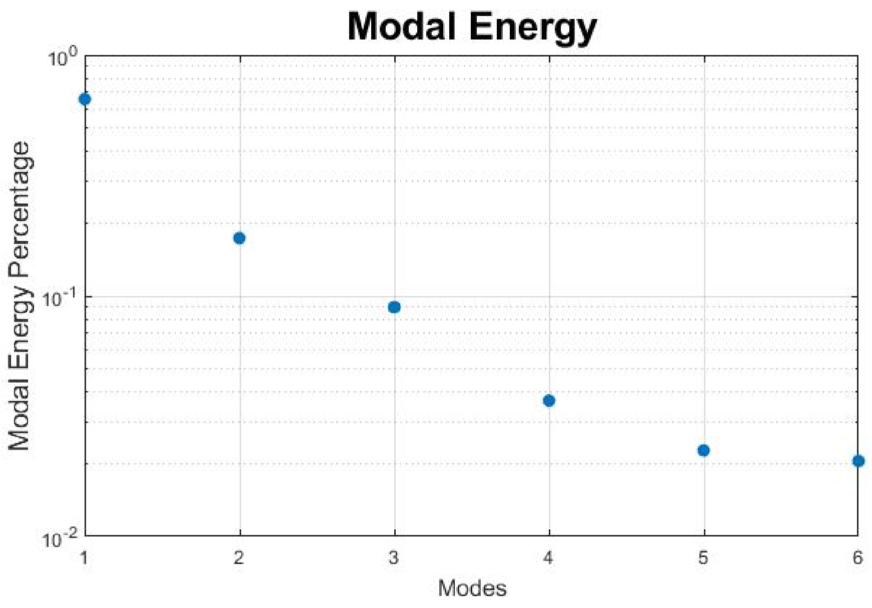

Figure 3: Modal Energies of problem 2

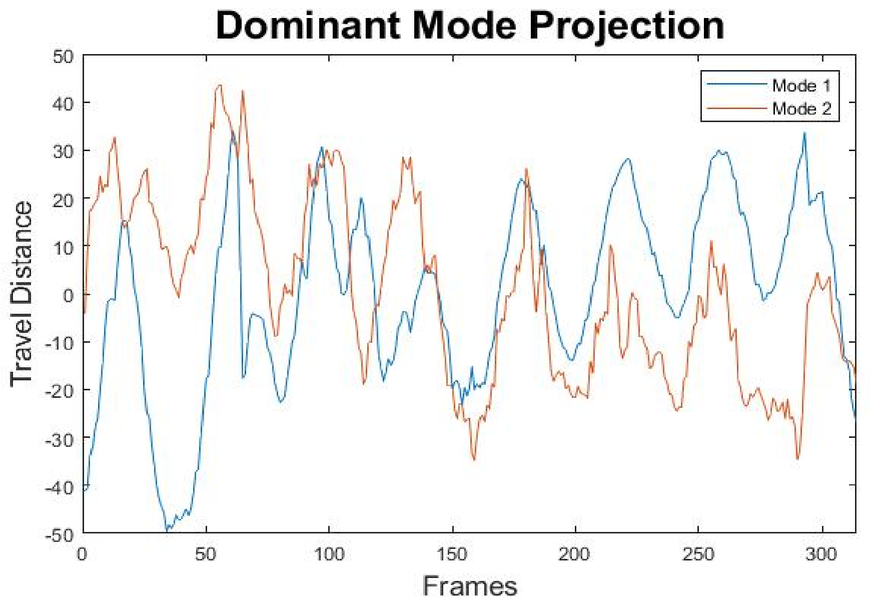

Figure 4: Modal projection of location data for problem 2

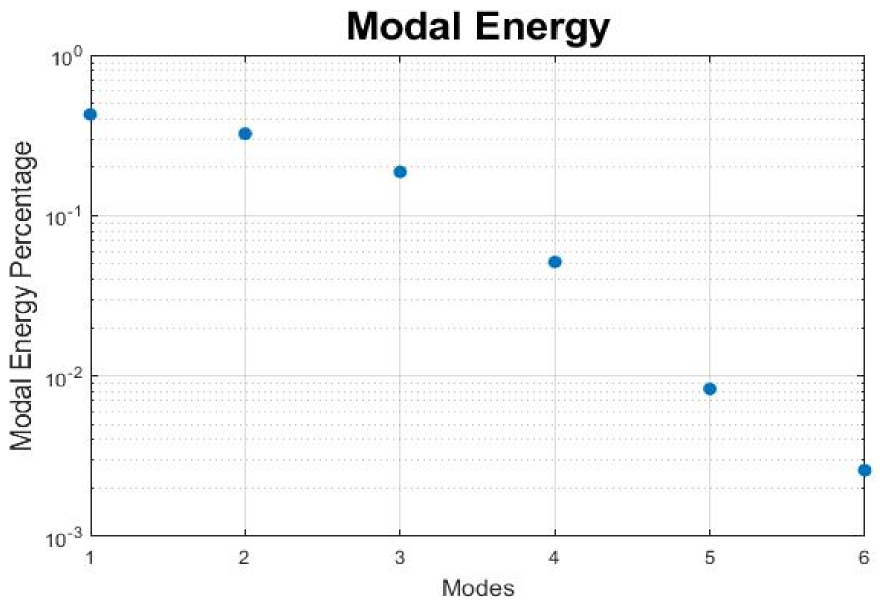

Figure 5: Modal Energies of problem 3

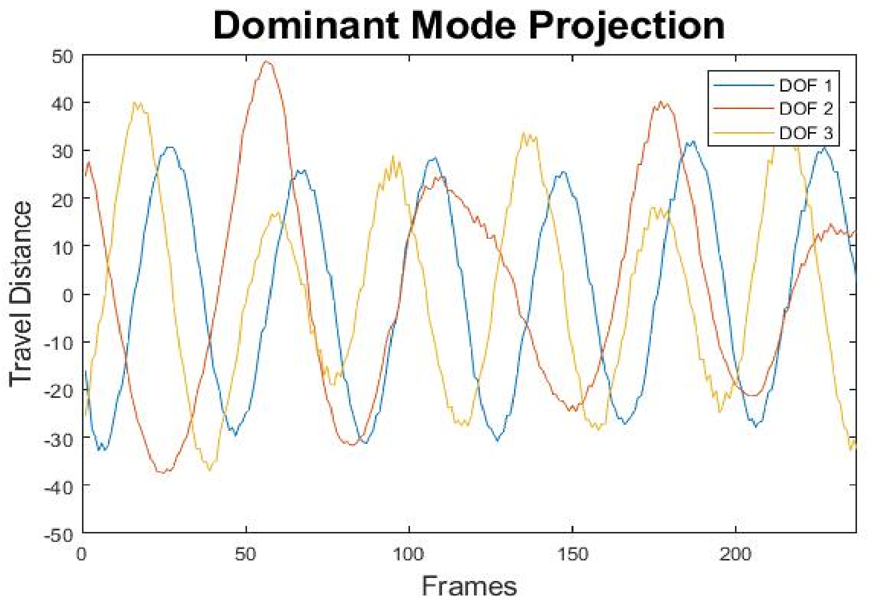

Figure 6: Modal projection of location data for problem 3

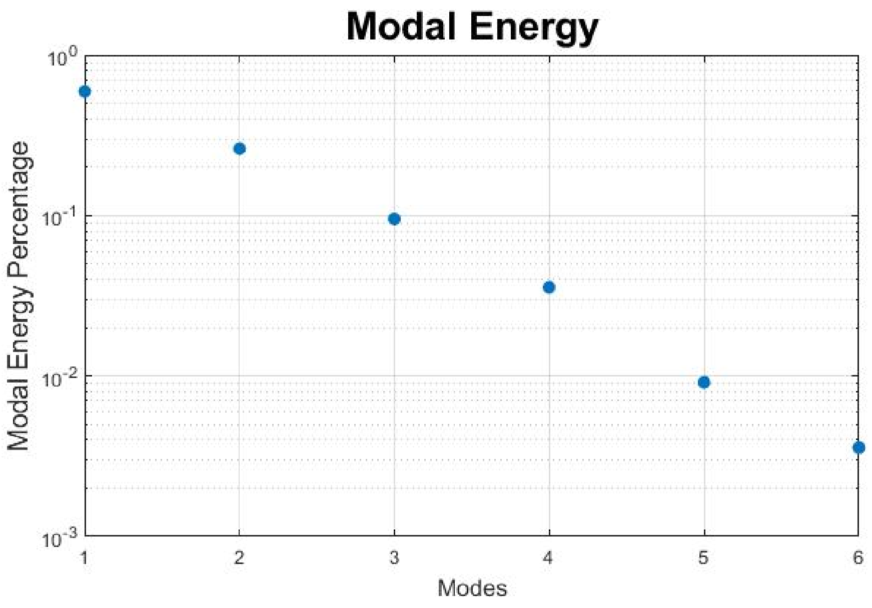

Figure 7: Modal Energies of problem 4

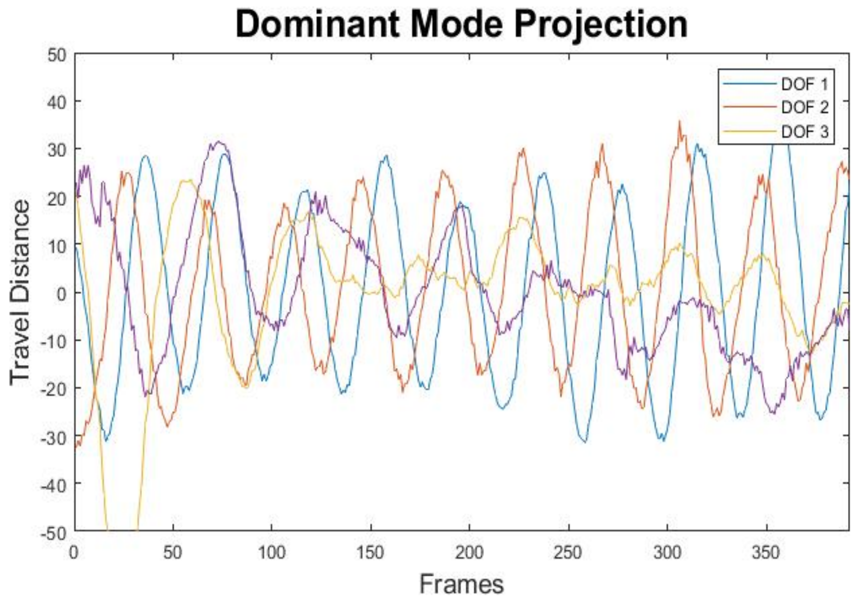

Figure 8: Modal projection of location data for problem 4

B MATLAB Functions

- `I = rgb2gray(RGB)` converts the truecolor image RGB to the grayscale image I. The rgb2gray function converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance. If you have Parallel Computing Toolbox™ installed, rgb2gray can perform this conversion on a GPU.

- `k = find(X)` returns a vector containing the linear indices of each nonzero element in array X. If X is a vector, then find returns a vector with the same orientation as X. If X is a multidimensional array, then find returns a column vector of the linear indices of the result.

- `TF = isnan(A)` returns a logical array containing 1 (true) where the elements of A are NaN, and 0 (false) where they are not. If A contains complex numbers, isnan(A) contains 1 for elements with either real or imaginary part is NaN, and 0 for elements where both real and imaginary parts are not NaN.

- `M = mean(A)` returns the mean of the elements of A along the first array dimension whose size does not equal 1. If A is a vector, then mean(A) returns the mean of the elements. If A is a matrix, then mean(A) returns a row vector containing the mean of each column. If A is a multidimensional array, then mean(A) operates along the first array dimension whose size does not equal 1, treating the elements as vectors. This dimension becomes 1 while the sizes of all other dimensions remain the same.

- `B = repmat(A,r1,...,rN)` specifies a list of scalars, r1,..,rN, that describes how copies of A are arranged in each dimension. When A has N dimensions, the size of B is size(A).*[r1...rN]. For example, repmat([1 2; 3 4],2,3) returns a 4-by-6 matrix.

- `[U,S,V] = svd(A,'econ')` produces an economy-size decomposition of m-by-n matrix A: m ¿ n — Only the first n columns of U are computed, and S is n-by-n. m = n — svd(A,'econ') is equivalent to svd(A). m ¡ n — Only the first m columns of V are computed, and S is m-by-m. The economy-size decomposition removes extra rows or columns of zeros from the diagonal matrix of singular values, S, along with the columns in either U or V that multiply those zeros in the expression A = U*S*V'. Removing these zeros and columns can improve execution time and reduce storage requirements without compromising the accuracy of the decomposition.

- `semilogy(X,Y)` plots x- and y-coordinates using a linear scale on the x-axis and a base-10 logarithmic scale on the y-axis. To plot a set of coordinates connected by line segments, specify X and Y as vectors of the same length. To plot multiple sets of coordinates on the same set of axes, specify at least one of X or Y as a matrix.

C MATLAB Code

```
%% Importing Movie data
camera_1_data = load('cam1_1.mat');

camera_1_data.vidFrames1_2 = double(camera_1_data.vidFrames1_1);
[l w d numFrames1_1] = size(camera_1_data.vidFrames1_1);
%resolution and frame data for movie file
for k = 1:numFrames1_1
    mov1_1(k).cdata = uint8(camera_1_data.vidFrames1_1(:,:,:,k));
    mov1_1(k).colormap = [];
end

for j = 1:numFrames1_1
   X_1_1 = rgb2gray(frame2im(mov1_1(j)));
   %converting movie to b&w
   X1(:,:,j) = double(X_1_1);
   %casting movie data to a double precision number series
   %  imshow(X_1_2); drawnow;
end
```

Listing 1: Conversion of movie file to matrices

```
%Extraction of light location
XCrop = X1(250:480,:,:);
%Cropped frames of video containing only the light
for n = 1:numFrames1_2
    if n ==1
    [row col] = find(XCrop(:,:,n)>=240);
    %find location of white pixel values
    for i = 1:length(row)
        row(i) = row(i) + 250;
    end
    %average X and Y location of pixels
    Arow_1(n) = sum(row)/length(row);
    Acol_1(n) = sum(col)/length(col);
    else %bounding box algorithm
        [row col]=find(X1(:,:,n)>=240);
        i = 1;
        while i <= length(row)
            if row(i)<(Arow_1(n-1)-30)|row(i)>(Arow_1(n-1)+30)
                row(i)=[]; col(i)=[];
            else
                i = i+1;
            end
        end
        i = 1;
         while i <= length(row)
            if col(i)<(Acol_1(n-1)-30)|col(i)>(Acol_1(n-1)+30)
                row(i)=[]; col(i)=[];
            else
                i = i+1;
            end
        end
        Arow_1(n) = sum(row)/length(row);
        Acol_1(n) = sum(col)/length(col);
    end
end
%NaN removal
Arow_2(isnan(Arow_2))=0;
Acol_2(isnan(Acol_2))=0;
%linar interpolation
for i = 1:length(Arow_2)
    if Arow_2(i)==0
        Arow_2(i) = (Arow_2(i-1) + Arow_2(i+1))/2;
        Acol_2(i) = (Acol_2(i-1) + Acol_2(i+1))/2;
    end
end
```

Listing 2: Pixel search algorithm used to extract location data for the moving mass

```
 %% PCA Analysis
X = [Acol_1;Arow_1;Acol_2;Arow_2;Acol_3;Arow_3]; %data matrix
[m,n]=size(X); %size of X
Xave = mean(X,2);
Xm=X-repmat(Xave,1,n); %subtract mean of data
% scatter3(Xm(1,:),Xm(2,:),Xm(3,:));

[u,s,v]=svd(Xm'/sqrt(n-1),'econ'); % perform the SVD
%modal energy plot
figure(2)
semilogy(s/sum(s),'o','MarkerFaceColor',[0 0.447 0.741]);
grid on
title('Modal Energy');

Y=u.*X'; % produce the principal components projection
figure(3)
subplot(2,1,1);
plot(u(:,1));
hold on
plot(u(:,2));
subplot(2,1,2);
plot(Y(:,1:2));
```

Listing 3: Principle component analysis of the Data matrix