AMATH 482 - Assignment 3

Clarisa Leu-Rodriguez

Winter 2021 - February 24th, 2021

Abstract

In this paper, we explore a dataset containing video data from four different oscillation scenarios of a spring-mass system. Each scenario was filmed from three cameras, each in different locations/from different angles. Our goal is to illustrate various aspects of Principal Component Analysis (PCA), its practical usefulness, and the effects of noise on PCA algorithms through performing the PCA method on the given dataset. We explore the ideas of video/image filtering and principal component analysis in this paper - as well as provide directions for future work on this problem and advantages/disadvantages of PCA.

Section I. Introduction and Overview

We are given a dataset containing video data from four different oscillation scenarios of a spring-mass system. Each scenario is filmed from three different cameras, each from a different position/from a different angle. The spring-mass system consists of a paint can (mass) attached to a stretchy cord (spring), where the paint can also has an attached light on the top of it. Our goal when analyzing these datasets is to perform the PCA method in order to reduce the dimensionality of the spring-mass system (in order to remove redundant data) and perform a low rank approximation of the motion of the mass. In order to use PCA on the given data, for each scenario we first extract the mass's position from the video frames through the use of image processing.

The first scenario is the ideal case, where the entire motion of the system is in the z-direction, where a simple harmonic motion can be observed. The second scenario is the noisy case, where we observe the same oscillation from the first scenario but the camera is shaking when recording the video. The third scenario observes the same spring-mass system, but with horizontal displacement introduced into the system - where the spring-mass system is released off-center as to produce motion both in the z-direction and x-y plane. The fourth scenario is the case with horizontal displacement and rotation, where the spring-mass system is released off-center and also rotates in order to produce motion both in the z-direction and x-y plane. In Figure 1 below, we show the different angles/positions of the cameras for the first scenario - although these angles/positions are mostly held constant for all of the scenarios (with the exception of the second scenario - where the camera is shaking) with a margin of error.

Figure 1: Pictures of the spring-mass systems from the different cameras (which were each taken from different positions/at different angles) for the first dataset (ideal case). Figure 1a (left most figure) corresponds to cam1_1.mat, Figure 1b (middle figure) corresponds to cam2_1.mat, and Figure 1c (right most figure) corresponds to cam3_1.mat from the given dataset.

Section II. Theoretical Background

PCA is the process of computing the principal components of a system and using them to perform a change of basis on the data - reducing the dimensionality of the original dataset and performing a low rank approximation of the original system. That is, PCA allows us to determine the lowest number of principal components needed to appropriately summarize a given dataset as well as those principal components without any additional knowledge as to how the system behaves. In this section, I will discuss the mathematics and theory behind PCA as well as to how it applies to our solution.

Section II.I. Standardization & Scaling Data to Reduce Bias in PCA

Before performing the PCA method on our dataset, we standardize our dataset to ensure that all of our data contributes equally to our analysis as PCA is biased towards features with high variance as it assumes there is only one common, shared variance in the dataset. That is, if there are large differences between the ranges of our variables in our data, those variables with larger ranges will dominate over those with small ranges, which leads to biased results. Standardizing transforms our data to comparable scales and helps to prevent this bias in values with large variation. Recall that standardizing involves subtracting the mean of the dataset from each datapoint and dividing by the standard deviation of the dataset. The formula is:

$$z = \frac{x_i - \mu}{\sigma}$$

Equation 1: The standardization of x where μ is the mean of x and σ is the standard deviation of x.

In our solution, after extracting the time series data (where this portion is discussed in the Section III. Algorithm Implementation and Development), we subtract the mean from our data as to reduce bias in data points with large variation and divide by the square root of the number of data points in our data set to better align our data in time as the video data from each of the cameras are not synchronized. While we did not standardize our solution and instead did this method of scaling, we certainly could try to standardize our data beforehand in the future & compare our results. It is important to note that when performing PCA - you should choose to scale or not scale your data according to the type of data collected/its context (i.e. - sometimes scaling your data is not necessary or and the way you scale your dataset/standardize it might require trial and error).

Section II.II. Singular Value Decomposition (SVD) & Energy

SVD is the factorization of a matrix which is a generalization of the eigendecomposition of a square normal matrix to any m x n matrix - where every m x n matrix A has a SVD. The SVD of a matrix A is shown below.

$$A = U\Sigma V^*$$

Equation 2: The SVD of a Matrix A where A is m x n.

In Equation 2, if A is complex - U is an m x m complex unitary matrix (i.e. its conjugate transpose is also it's inverse - $U^*U = UU^* = I$), Σ is an m x n rectangular diagonal matrix with non-negative, real number along its diagonal, and V^* is an n x n complex unitary matrix. If A is real, then U and $V^{\mathrm{T}} = V^*$ are real orthogonal matrices (i.e. their columns and rows are orthonormal vectors - $U^{\mathrm{T}}U = UU^{\mathrm{T}} = I$ and $V^{\mathrm{T}}V = V V^{\mathrm{T}} = I$).

U and V are rotational matrices and Σ is a stretching matrix - where the diagonal entries along Σ are the uniquely determined singular values of A, the columns of U are left-singular vectors, and the columns of V are right-singular vectors of A. The total number of non-zero singular values gives us the rank of A - where the SVD is often used as the preferred method for matrix rank calculations in mathematical software, as often it is more reliable due to approximation errors in real numbers. Additionally, other properties of the SVD include:

- The singular values of A are always ordered from largest to smallest (i.e. $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n \geq 0$) along the diagonals of Σ.
- Letting $r = \mathrm{rank}(A)$, we have that $\{u_1, u_2, ..., u_r\}$ is a basis for the range of A and $\{v_r + 1, ..., v_n\}$ is a basis for the null space of A.

As it pertains to PCA, the SVD of a matrix tells us the dimension and the direction of the principal components of the matrix. It reveals information as to how much data is in each dimension/rank approximation (via the singular values of A). In our solution,

after determining the principal components in our system - we then determine which ones contribute the most energy to our system through looking at the energy captured in each dimension/rank through the following equation:

$$energy_n = \frac{\sigma_1^2+...+\sigma_n^2}{\sigma_1^2+...+\sigma_r^2}$$

Equation 3: The energy contained in the rank-n approximation of A (where A is of rank r) captured by the singular values (i.e. the $\boldsymbol{\sigma}_i$'s) of A.

When analyzing which principal components contribute the most to our system in the PCA method, we see which PCA modes have the highest amount of energy captured in its dimension. This tells us which other modes are also redundant to summarizing the system (i.e. - those with low amounts of energy captured with respect to the other modes). This is what we did in our solution, as well as looking at how well our low rank approximation of our system did by projecting our original data back onto the principal component bases to see how well our low-rank approximations did when compared to the original data by trying to match features seen back to our original positional data.

Section III. Algorithm Implementation and Development

To start this problem, we first import the given video data into MATLAB - where we have video data from 3 different cameras capturing 4 different spring oscillation scenarios (i.e. 12 videos are given which are represented as MATLAB matrices - where the frame width is 640 pixels, frame height is 480 pixels, the frame length for the videos varies between the videos, and each of the frames in the videos contain a true RGB value).

In order to perform PCA on the spring-mass system, the first major task is to determine the positional coordinates of the mass (i.e. the paint can) over time. This was done by looking at the frames in each video and performing image processing on the video frames. That is, for each frame in our video - we first convert the image to grayscale. This is because as shown in Figure 1, the paint can is white and there is also a light attached to the top of the can which helps us track the can. Recall that when an image is represented in grayscale, all of the pixels represent different shades of gray - where 0 is black and 255 is white. By looking at the image in grayscale - this allows us to easily pick out the light or white color on the paint can by looking at pixel values greater than a certain value (i.e. shades of gray closer to 255 which is white).

In addition to transforming the image of each video frame to grayscale, we also attempted to crop out the background "noise" of the room - which also contained white colors or reflections of light (in Figure 1a, this can be seen in the chair's seat and in Figure 1c, can be seen with the light reflecting off of the whiteboard). This was done by setting all of the values outside a certain x and y range to zero - and amplifying values inside this range. This range of x and y is where we determined the paint can was by watching each respective video. After cropping the image - we amplify the white spots on the can and the light on the paint cans lid by applying a threshold to the grayscale pixel values (where the threshold filters out values which are less than this threshold). After applying this threshold - we average the remaining positional coordinates which will represent the positional coordinates of the can at that time frame which is used for PCA. This entire process of filtering each video frame is also illustrated in Figure 2 below. Through the use of cropping out noise in each video frame (i.e. only looking at the spring-mass system), filtering out pixels below a certain grayscale value threshold, and averaging the positional coordinates in the x and y direction - we were able to gather time series data on the paint cans position over time for all of the scenarios.

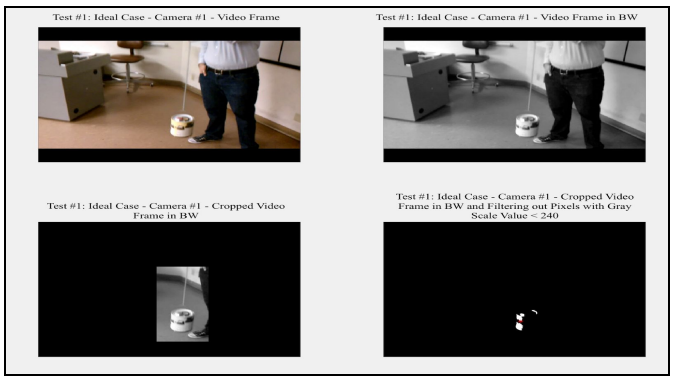

Figure 2: Images showing the process used to isolate the paint cans positional coordinates over time for a single video frame for scenario one (camera one). Figure 1a (upper left figure) shows the original video frame, Figure 1b (upper right figure) shows the original video frame in grayscale, Figure 1c (bottom left figure) shows the cropped grayscale image as to filter out background noise, and Figure 1d (bottom left corner) shows the filtering of the image data according to looking a grayscale pixel values greater than 240. The red dot in Figure 1d also shows the averaged position which is used for PCA.

Additionally, before performing PCA we combine all of the positional data gathered for each camera, for each scenario into a single matrix. As the beginning/start of the oscillation varied between different videos of the same oscillation scenario - we also attempted to better align the videos in time regarding the start/stop of the oscillation. This was done by looking at the lowest y-coordinate within the first however many frames (determined through trial and error - and varied with each scenario) and trimmed each of the videos to start at this frame number. In addition, as the number of frames/length of each video varied, we trim the positional data according to the video with the smallest number of frames. The combined positional data used for PCA was of dimension 6 x min_frame_num.

With our combined positional data, we then try to reduce any bias in the data before performing SVD on the matrix by subtracting the mean from all of the rows of our positional data and dividing by the square root of the number data points to better align in time as well. After scaling the data, we perform SVD on the matrix, analyze the energies captured in each mode/dimension to determine which ones contribute the most/least to the system, as well as look at how well our low-rank approximation did when compared to our original positional data.

Section IV. Computational Results

Test #1: Ideal Case	0.8889	0.1008	0.0053	0.0024	0.0018	0.0008
Test #2: Noisy Case	0.652	0.1642	0.1082	0.0334	0.0216	0.0207
Test #3: Horizontal Displacement	0.5994	0.2598	0.1044	0.0264	0.0063	0.0037
Test #4: Horizontal Displacement & Rotation	0.6696	0.1991	0.0936	0.0287	0.0059	0.0032

Table 1: The energies captured for each rank-n approximation for $n = 1, ..., 6$ for each test case. Reading from left to right, column 1 corresponds to the scenario, column 2 corresponds to the energy captured for rank 1 and the last column corresponds to the energy captured for rank 6.

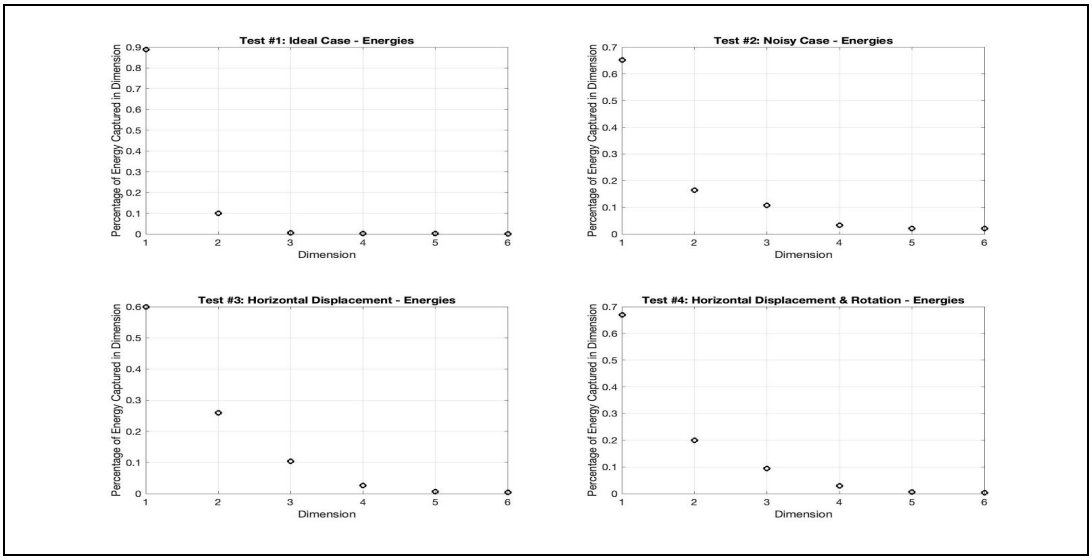

Figure 3: Figure showing the different energies captured for each scenario using Equation 3.

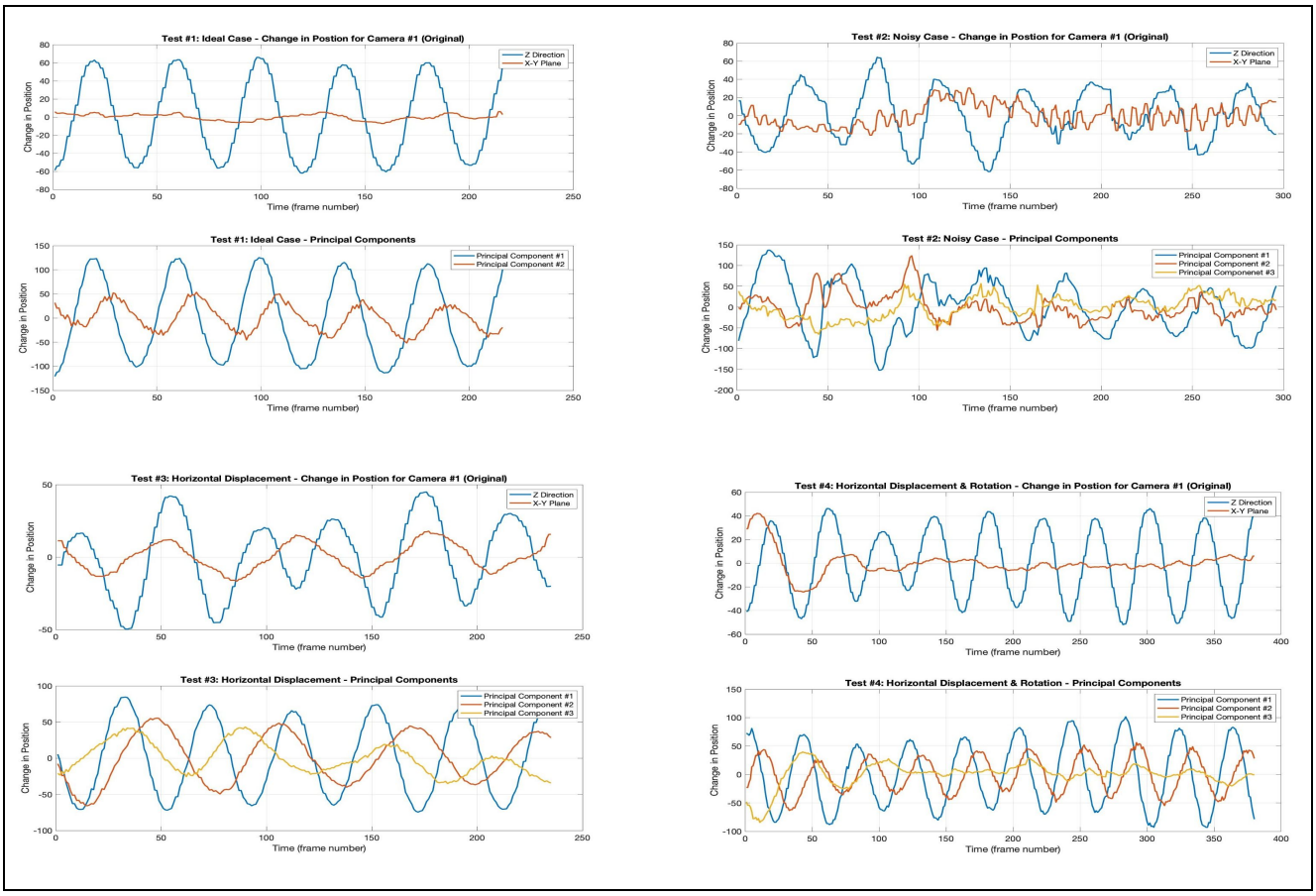

Figure 4: Figure showing the original positional data of the mass for each scenario as well as our original data projected on the principal components bases which we determined to contribute the most to the overall systems motion by looking at the energies in Table 1 and Figure 3. Figure 4a corresponds to the two uppermost figures on the left, Figure 4b corresponds to the two uppermost figures on the right, Figure 4c corresponds to the two bottom most figures on the left, and Figure 4d corresponds to the two bottom most figures on the right.

Section V. Summary and Conclusions

Through the use of image processing and principal component analysis (PCA), we were able to reduce the dimensionality of the original positional data of the mass in the spring-mass system and perform a low-rank approximation of the system.

Analyzing the first test's results - we see that most of the energy is captured in the first dimension at 88.89% of the total energy and 10.08% of the energy being captured in the second dimension (from Table 1). The energies quickly drop off after the second dimension - implying the other dimensions are not very important when capturing the features of the original data. Looking at Figure 4a - we plot the original positional data we gathered on the paint can. We see an oscillation in the z-direction and not much movement in the x-y plane which matches what we would expect for the first scenario. We also plot our positional data projected onto the basis of the first and second principal components in Figure 4a. Looking at how well our PCA did - we were able to capture the mass's motion in the spring-mass system fairly well with a one principal component, although some of the motion is also captured in the second principal component as well which is shown.

Analyzing the second test's results - we see that most of the energy is captured in the first dimension at 65.2% of the total energy. We also see that the second and third dimension each capture 16.42% and 10.82%, respectively. While there is a large drop off from the first dimension to the second dimension, I would say that three principal components would be needed to represent this system. While the second test represents the same simple harmonic motion of the mass, the added noise from the shaky camera makes it so additional principal components are needed to capture the noisy features of the data. Looking at Figure

4b, we see that our original data shows the shape of a simple oscillation as in the first scenario - although there is additional noise to the oscillation in both the *z*-direction and *x*-*y* plane which was also captured. Additionally - our principal components capture this noise in our approximation which can be seen in Figure 4b.

Analyzing the third test's results, we see that like in the second scenario, most of the systems energies is captured in the first three dimensions which can be seen in Table #1. The first dimension accounts for 59.94% total energy of the system, the second dimension accounts for 16.42% total energy of the system, and the third dimension accounts for 10.44% total energy of the system. Compared to scenario one and two, this one varies from them as a horizontal displacement was introduced to the system. Looking at Figure 4c, the original positional data gathered shows this added horizontal motion in the *x*-*y* plane clearly and the oscillation of the mass in the *z*-direction. Looking at our original data projected onto the basis of the first three principal components, while not perfect, PCA did a fair job in estimating the mass's motion in the scenario.

Analyzing the fourth test's results, most of the systems energies are also captured in the first three dimensions. The first dimension accounts for 66.96% total energy of the system, the second dimension accounts for 19.91% total energy of the system, and the third dimension accounts for 9.36% total energy of the system. The fourth test involved not only adding a horizontal displacement to the ideal case but also a rotation to the paint can. Looking at Figure 4d (lower right hand figure in Figure 4), we plot the gathered positional data of the paint can over time. In the original positional data - we see an oscillation in the *z*-direction of the mass and in the *x*-*y* plane we also see some motion - although it is not necessarily as apparent what this motion is doing (i.e. if it is capturing a rotation or horizontal displacement). Looking at our original data projected onto the basis of the first three principal components, we see that PCA does a fairly good job at capturing the oscillation in the *z*-direction and the horizontal displacement in the *x*-*y* plane - although it is hard to determine if it captured the rotation of the can well as matching the principal component back to the motion of the can is difficult in this scenario.

Future work for this problem would be to try to improve the approximations of the position of the paint cans coordinates to improve our results from PCA, as the better our observations are of the true motion of the can - the better PCA will summarize the system. In addition - it is also important to note that this problem was simplified in a few ways - some being that we were told how the original system behaved and knew what to look for in our PCA/how to approach the given dataset. In most problems involving real world data of sufficient complexity, this information is sometimes unknown to us and the preprocessing of the data/analysis of the principal components would generally be more difficult without this prior knowledge. In addition, while we saw that PCA did very well in removing correlated features in our dataset taken from the three different cameras and provided a generally good low rank approximation of the system without overfitting the data - PCA still has some disadvantages. As PCA turns features of a given system into principal components (linear combinations of the original features of the dataset), these components are not easily interpreted in more complicated systems. This was especially true for scenarios two through four - as it was hard to distinguish which principal component contributed to which feature of the original data set to a large degree of certainty. Also, as mentioned previously, PCA is biased towards features with high variance, making it not possible to capture the true motion of a system if the given data contains redundant noise (this was shown in scenario two).

Section VI. References

- https://en.wikipedia.org/wiki/Singular_value_decomposition - I used this source to better understand Singular Value Decomposition.
- https://en.wikipedia.org/wiki/Principal_component_analysis - I used this source to better understand Principal Component Analysis.
- In addition, the course notes provided on the course website and all lecture material was used.

Appendix A. MATLAB Functions Used and Brief Implementation Explanation

This appendix contains a list of the MATLAB functions used and additional information regarding how it was used to solve the given problem. Additional information can also be found in Section II. Theoretical Background, Section III. Algorithm Implementation and Development, and in the comments of Appendix B. MATLAB Codes. In addition, the Image Processing Toolbox in MATLAB was downloaded for this paper in order to process the video frames of each video. Note, descriptions of functions were sourced from the MATLAB documentation which can be found here online: https://www.mathworks.com/help/matlab/.

- **load**(dat) - loads the given data (represented as a MAT-file) into the workspace. This was used to load the given dataset into MATLAB for analysis.
- M = **size**(X, DIM1, DIM2, …, DIMN) returns the lengths of the dimensions DIM1, …, DIMN as a row vector. This was used to determine the sizes of different dimensions in matrices.
- [X, MAP] = **frame2im**(F) returns the indexed image X and associated colormap MAP from the single movie frame F. If the frame contains true-color data, the M x N x 3 matrix MAP is empty. We used this to convert each video frame to its respective image data in order to process the frame in MATLAB.
- I = **rgb2gray**(RGB) converts the true color image RGB to the grayscale intensity image I. This was used to convert each video frame's image data to the grayscale.
- I = **find**(X) returns the linear indices corresponding to the nonzero entries of the array X. X may be a logical expression. We used this to find where in our cropped grayscale image the grayscale values were greater than a certain grayscale value (in order to amplify values closer to 255).
- [I, J] = **ind2sub**(SIZ, IND) returns the arrays I and J containing the equivalent row and column subscripts corresponding to the index matrix IND for a matrix of size SIZ. We used this to locate the positions of the indices found from find in our filtered data within the video frame.
- **imshow**(X) displays the image X in MATLAB. We used this to display the image for each video frame in our data.
- **mean**(X, DIM) takes the mean along the dimension DIM of X. This was used to calculate the mean of our positional data when scaling our combined positional data and also when averaging our positional coordinates in get_time_series_dat.
- B = **repmat**(A, M, N) creates a large matrix B consisting of an M-by-N tiling of copies of A. If A is a matrix, the size of B is [size(A, 1) * M, size(A, 2) * N]. This was used in order to subtract the mean from our positional data when scaling.
- [U, S, V] = **svd**(X) produces a diagonal matrix S, of the same dimension as X and with nonnegative diagonal elements in decreasing order, and unitary matrices U and V so that X = U * S * V'. This was used to calculate the SVD of our scaled positional data.
- **zeros**(M, N, P, ...) generates an M-by-N-by-P-by-... array of zeros. We use the zeros function in MATLAB to preallocate memory before computation - which saves the computer time as it doesn't need to grab more buckets during computation as the space to store the result has already been preallocated.

Appendix B. MATLAB Codes

This appendix contains the MATLAB code used in this paper. Please refer to Appendix A. MATLAB Functions Used and Brief Implementation Explanation for more information regarding how each of the MATLAB functions used in our code works, as well as the process used to solve the given problem.

`assignment_3.m`

```
%{
    Assignment #3 - PCA and a Spring-Mass System
    AMATH482 - Computational Methods For Data Science - Feb. 24th, 2021
    Taught by Professor Jason J. Bramburger (Winter 2021)
    Written By: Clarisa Leu-Rodriguez - email: cleu@uw.edu
%}

%% Import Data

clear all; close all; clc;

% Test #1: Ideal Case
load('cam1_1.mat');
load('cam2_1.mat');
load('cam3_1.mat');

% Test #2: Noisy Case
load('cam1_2.mat');
load('cam2_2.mat');
load('cam3_2.mat');

% Test #3: Horizontal Displacement
load('cam1_3.mat');
load('cam2_3.mat');
load('cam3_3.mat');

% Test #4: Horizontal Displacement and Rotation
load('cam1_4.mat');
load('cam2_4.mat');
load('cam3_4.mat');

% Constants used throughout script:
% Video data is of dimensions 480 x 640 x 3 x num_frames
% That is - the width of each frame is 640 pixels, the height is 480
% pixels, the pixels have an associated RGB value, and the number of frames
% vary between each set of data.
frame_width = size(vidFrames1_1, 2);
frame_height = size(vidFrames1_1, 1);  % Frame width and height are constant
                                       % between all datasets - however
                                       % frame width varies.


%% Test #1: Ideal Case - Extract Time Series Data
% Find filter which crops out background well through trial/error - in addition
% to good value to set as the threshold for grayscale values to look at.

filter_cam_1 = zeros(frame_height, frame_width);
% Specify x-bound/y-bound to keep - these are the bounds for the paint can
% in the video.
x_bound_filt_1 = 280:1:400;
```

```
y_bound_filt_1 = 140:1:450;
filter_cam_1(y_bound_filt_1, x_bound_filt_1) = 1;
time_series_dat_cam_1 = get_time_series_dat(vidFrames1_1, filter_cam_1, 240, 0);

filter_cam_2 = zeros(frame_height, frame_width);
% Specify x-bound/y-bound to keep - these are the bounds for the paint can
% in the video.
x_bound_filt_2 = 230:1:360;
y_bound_filt_2 = 80:1:405;
filter_cam_2(y_bound_filt_2, x_bound_filt_2) = 1;
time_series_dat_cam_2 = get_time_series_dat(vidFrames2_1, filter_cam_2, 240, 0);

filter_cam_3 = zeros(frame_height, frame_width);
% Specify x-bound/y-bound to keep - these are the bounds for the paint can
% in the video.
x_bound_filt_3 = 220:1:500;
y_bound_filt_3 = 220:1:380;
filter_cam_3(y_bound_filt_3, x_bound_filt_3) = 1;
time_series_dat_cam_3 = get_time_series_dat(vidFrames3_1, filter_cam_3, 240, 0);


%% Test #1: Ideal Case - Perform PCA
% Data should be of the same dimension for PCA - so align data.
dat = align_dat(time_series_dat_cam_1, time_series_dat_cam_2, time_series_dat_cam_3, 21);
% Subtract the mean from each row.
mean_val = mean(dat, 2);
dat = dat - repmat(mean_val, 1, size(dat, 2));

% Calculate SVD & Energies. Also scale by 1 / sqrt(# of time points) to align in
% space).
[U, S, V] = svd(dat' ./ sqrt(size(dat, 2)));
lambdas = diag(S).^2;
energies_1 = lambdas ./ sum(lambdas);  % Save energies to plot later.

% Project principal components to compare with original.
Y = dat' * V;


%% Test #1: Ideal Case - Figures
% Plot change in position from original data.
figure();
subplot(2, 1, 1);
x_axis = 1:size(dat, 2);
plot(x_axis, dat(2, :), x_axis, dat(1, :), 'Linewidth', 2);
ylabel("Change in Position"); xlabel("Time (frame number)");
title("Test #1: Ideal Case - Change in Position for Camera #1 (Original)");
legend("Z Direction", "X-Y Plane");
grid on; set(gca, 'Fontsize', 13);

% Plot projection.
subplot(2, 1, 2);
plot(x_axis, Y(:, 1), x_axis, Y(:, 2), 'Linewidth', 2);
ylabel("Change in Position"); xlabel("Time (frame number)");
title("Test #1: Ideal Case - Principal Components");
legend("Principal Component #1", "Principal Component #2");
grid on; set(gca, 'Fontsize', 13);
```

```
%% Test #2: Noisy Case - Extract Time Series Data
% Find filter which crops out background well through trial/error - in addition
% to good value to set as the threshold for grayscale values to look at.
filter_cam_1 = zeros(frame_height, frame_width);
% Specify x-bound/y-bound to keep - these are the bounds for the paint can
% in the video.
x_bound_filt_1 = 290:1:415;
y_bound_filt_1 = 160:1:425;
filter_cam_1(y_bound_filt_1, x_bound_filt_1) = 1;
time_series_dat_cam_1 = get_time_series_dat(vidFrames1_2, filter_cam_1, 240, 0);

filter_cam_2 = zeros(frame_height, frame_width);
% Specify x-bound/y-bound to keep - these are the bounds for the paint can
% in the video.
x_bound_filt_2 = 165:1:425;
y_bound_filt_2 = 50:1:475;
filter_cam_2(y_bound_filt_2, x_bound_filt_2) = 1;
time_series_dat_cam_2 = get_time_series_dat(vidFrames2_2, filter_cam_2, 240, 0);


filter_cam_3 = zeros(frame_height, frame_width);
% Specify x-bound/y-bound to keep - these are the bounds for the paint can
% in the video.
x_bound_filt_3 = 235:1:495;
y_bound_filt_3 = 200:1:380;
filter_cam_3(y_bound_filt_3, x_bound_filt_3) = 1;
time_series_dat_cam_3 = get_time_series_dat(vidFrames3_2, filter_cam_3, 240, 0);


%% Test #2: Noisy Case - Perform PCA
% Data should be of the same dimension for PCA - so align data.
dat = align_dat(time_series_dat_cam_1, time_series_dat_cam_2, time_series_dat_cam_3, 19);
% Subtract the mean from each row.
mean_val = mean(dat, 2);
dat = dat - repmat(mean_val, 1, size(dat, 2));

% Calculate SVD & Energies. Also scale by 1 / sqrt(# of time points) to align in
% space).
[U, S, V] = svd(dat' ./ sqrt(size(dat, 2)));
lambdas = diag(S).^2;
energies_2 = lambdas ./ sum(lambdas);  % Save energies to plot later.

% Project principal components to compare with original.
Y = dat' * V;


%% Test #2: Noisy Case - Figures
% Plot change in position from original data.
figure();
subplot(2, 1, 1);
x_axis = 1:size(dat, 2);
plot(x_axis, dat(2, :), x_axis, dat(1, :), 'Linewidth', 2);
ylabel("Change in Position"); xlabel("Time (frame number)");
title("Test #2: Noisy Case - Change in Position for Camera #1 (Original)");
legend("Z Direction", "X-Y Plane");
grid on; set(gca, 'Fontsize', 13);

% Plot projection.
```

```
subplot(2, 1, 2);
plot(x_axis, Y(:, 1), x_axis, Y(:, 2), x_axis, Y(:, 3), ...
        'Linewidth', 2);
ylabel("Change in Position"); xlabel("Time (frame number)");
title("Test #2: Noisy Case - Principal Components");
legend("Principal Component #1", "Principal Component #2", ...
    "Principal Component #3");
grid on; set(gca, 'Fontsize', 13);


%% Test #3: Horizontal Displacement - Extract Time Series Data
% Find filter which crops out background well through trial/error - in addition
% to good value to set as the threshold for grayscale values to look at.

filter_cam_1 = zeros(frame_height, frame_width);
% Specify x-bound/y-bound to keep - these are the bounds for the paint can
% in the video.
x_bound_filt_1 = 265:1:415;
y_bound_filt_1 = 225:1:440;
filter_cam_1(y_bound_filt_1, x_bound_filt_1) = 1;
time_series_dat_cam_1 = get_time_series_dat(vidFrames1_3, filter_cam_1, 240, 0);

filter_cam_2 = zeros(frame_height, frame_width);
% Specify x-bound/y-bound to keep - these are the bounds for the paint can
% in the video.
x_bound_filt_2 = 165:1:425;
y_bound_filt_2 = 140:1:425;
filter_cam_2(y_bound_filt_2, x_bound_filt_2) = 1;
time_series_dat_cam_2 = get_time_series_dat(vidFrames2_3, filter_cam_2, 250, 0);

filter_cam_3 = zeros(frame_height, frame_width);
% Specify x-bound/y-bound to keep - these are the bounds for the paint can
% in the video.
x_bound_filt_3 = 225:1:490;
y_bound_filt_3 = 160:1:360;
filter_cam_3(y_bound_filt_3, x_bound_filt_3) = 1;
time_series_dat_cam_3 = get_time_series_dat(vidFrames3_3, filter_cam_3, 240, 0);


%% Test #3: Horizontal Displacement - Perform PCA
% Data should be of the same dimension for PCA - so align data.
dat = align_dat(time_series_dat_cam_1, time_series_dat_cam_2, time_series_dat_cam_3, 14);
% Subtract the mean from each row.
mean_val = mean(dat, 2);
dat = dat - repmat(mean_val, 1, size(dat, 2));

% Calculate SVD & Energies. Also scale by 1 / sqrt(# of time points) to align in
% space).
[U, S, V] = svd(dat' ./ sqrt(size(dat, 2)));
lambdas = diag(S).^2;
energies_3 = lambdas ./ sum(lambdas);  % Save energies to plot later.

% Project principal components to compare with original.
Y = dat' * V;


%% Test #3: Horizontal Displacement - Figures
% Plot change in position from original data.
```

```
figure();
subplot(2, 1, 1);
x_axis = 1:size(dat, 2);
plot(x_axis, dat(2, :), x_axis, dat(1, :), 'Linewidth', 2);
ylabel("Change in Position"); xlabel("Time (frame number)");
title("Test #3: Horizontal Displacement - Change in Position for Camera #1 (Original)");
legend("Z Direction", "X-Y Plane");
grid on; set(gca, 'Fontsize', 13);

% Plot projection.
subplot(2, 1, 2);
plot(x_axis, Y(:, 1), x_axis, Y(:, 2), x_axis, Y(:, 3), 'Linewidth', 2);
ylabel("Change in Position"); xlabel("Time (frame number)");
title("Test #3: Horizontal Displacement - Principal Components");
legend("Principal Component #1", "Principal Component #2", "Principal Component #3");
grid on; set(gca, 'Fontsize', 13);


%% Test #4: Horizontal Displacement & Rotation - Extract Time Series Data
% Find filter which crops out background well through trial/error - in addition
% to good value to set as the threshold for grayscale values to look at.

filter_cam_1 = zeros(frame_height, frame_width);
% Specify x-bound/y-bound to keep - these are the bounds for the paint can
% in the video.
x_bound_filt_1 = 275:1:470;
y_bound_filt_1 = 225:1:440;
filter_cam_1(y_bound_filt_1, x_bound_filt_1) = 1;
time_series_dat_cam_1 = get_time_series_dat(vidFrames1_4, filter_cam_1, 230, 0);

filter_cam_2 = zeros(frame_height, frame_width);
% Specify x-bound/y-bound to keep - these are the bounds for the paint can
% in the video.
x_bound_filt_2 = 175:1:450;
y_bound_filt_2 = 110:1:390;
filter_cam_2(y_bound_filt_2, x_bound_filt_2) = 1;
time_series_dat_cam_2 = get_time_series_dat(vidFrames2_4, filter_cam_2, 250, 0);

filter_cam_3 = zeros(frame_height, frame_width);
% Specify x-bound/y-bound to keep - these are the bounds for the paint can
% in the video.
x_bound_filt_3 = 245:1:490;
y_bound_filt_3 = 130:1:320;
filter_cam_3(y_bound_filt_3, x_bound_filt_3) = 1;
time_series_dat_cam_3 = get_time_series_dat(vidFrames3_4, filter_cam_3, 220, 0);


%% Test #4: Horizontal Displacement & Rotation - Perform PCA
% Data should be of the same dimension for PCA - so align data.
dat = align_dat(time_series_dat_cam_1, time_series_dat_cam_2, time_series_dat_cam_3, 15);
% Subtract the mean from each row.
mean_val = mean(dat, 2);
dat = dat - repmat(mean_val, 1, size(dat, 2));

% Calculate SVD & Energies. Also scale by 1 / sqrt(# of time points) to align in
% space).
[U, S, V] = svd(dat' ./ sqrt(size(dat, 2)));
lambdas = diag(S).^2;
```

```
energies_4 = lambdas ./ sum(lambdas);  % Save energies to plot later.

% Project principal components to compare with original.
Y = dat' * V;


%% Test #4: Horizontal Displacement & Rotation - Figures
% Plot change in position from original data.
figure();
subplot(2, 1, 1);
x_axis = 1:size(dat, 2);
plot(x_axis, dat(2, :), x_axis, dat(1, :), 'Linewidth', 2);
ylabel("Change in Position"); xlabel("Time (frame number)");
title("Test #4: Horizontal Displacement & Rotation - Change in Position for Camera #1
(Original)");
legend("Z Direction", "X-Y Plane");
grid on; set(gca, 'Fontsize', 13);

% Plot projection.
subplot(2, 1, 2);
plot(x_axis, Y(:, 1), x_axis, Y(:, 2), x_axis, Y(:, 3), 'Linewidth', 2);
ylabel("Change in Position"); xlabel("Time (frame number)");
title("Test #4: Horizontal Displacement & Rotation - Principal Components");
legend("Principal Component #1", "Principal Component #2", "Principal Component #3");
grid on; set(gca, 'Fontsize', 13);


%% Plot Energies For All Tests to Compare
figure();
x_axis = 1:6;
subplot(2, 2, 1);
plot(x_axis, energies_1, 'ko', 'LineWidth', 2);
xlabel("Dimension"); ylabel("Percentage of Energy Captured in Dimension");
title("Test #1: Ideal Case - Energies");
grid on; set(gca, 'Fontsize', 12);

subplot(2, 2, 2);
plot(x_axis, energies_2, 'ko', 'LineWidth', 2);
xlabel("Dimension"); ylabel("Percentage of Energy Captured in Dimension");
title("Test #2: Noisy Case - Energies");
grid on; set(gca, 'Fontsize', 12);

subplot(2, 2, 3);
plot(x_axis, energies_3, 'ko', 'LineWidth', 2);
xlabel("Dimension"); ylabel("Percentage of Energy Captured in Dimension");
title("Test #3: Horizontal Displacement - Energies");
grid on; set(gca, 'Fontsize', 12);

subplot(2, 2, 4);
plot(x_axis, energies_4, 'ko', 'LineWidth', 2);
xlabel("Dimension"); ylabel("Percentage of Energy Captured in Dimension");
title("Test #4: Horizontal Displacement & Rotation - Energies");
grid on; set(gca, 'Fontsize', 12);
```

get_time_series_dat.m

```
%{
    Helper function to gather the time series data of the paint can (i.e. the
    mass), given the video data of the spring-mass system.

    Takes in the video frames (vid_frames) data, a function to crop the
    video as to remove any background "noise", a gray_scale_val to filter
    the video by (value from 0 to 255) as to isolate the light on the paint can,
    and a boolean value play_vid where if asserted - plays the passed in
    video as well as shows the gray scale filtering/crop function in action.

    Outputs a matrix (which is of size num_frames by 2) containing the (x, y)
    coordinates of where we suspect the paint can to be located -
    based on the given crop function and gray scale value used to filter each frame by.

    * Note, you will need the image processing toolbox in MATLAB to use this
    function. *
%}

function [time_series_dat] = get_time_series_dat(vid_frames, ...
                                crop_vid_func, gray_scale_val, play_vid)
    num_frames = size (vid_frames, 4);
    time_series_dat = zeros(num_frames, 2);
    for i = 1:num_frames
        % For each frame, construct a struct with two fields:
        %   - cdata which is the image data stored as an array of uint8 values,
        %   - colormap which will be empty as the video frames contain
        %     true (RGB) images. This is so we can use the function
        %     frame2im - which will return the image data associated with the
        %     passed in the video frame.
        frame_dat.cdata = vid_frames(:, :, :, i);
        frame_dat.colormap = [];
        image_dat = frame2im(frame_dat);  % Get image data for frame.

        % Convert to grayscale as it is easier to track the light on
        % top of the paint can (i.e. - we set a threshold for grayscale values
        % to let in where there are 256 different shades of gray with 0 being
        % black and 255 being white).
        image_dat_bw = rgb2gray(image_dat);

        % Pick threshold which looks best at picking out the light on top
        % of the paint can. In addition, crop the background of the video
        % with passed in filter function.
        cropped_dat = double(image_dat_bw).*crop_vid_func;
        filtered_dat = cropped_dat > gray_scale_val;

        % Find coordinates of bright spots - average them and set return
        % value which corresponds to where we suspect the can to be for this
        % slice in time. Averaging should be okay here if crop function
        % correctly crops out any background light reflected by the chair,
        % etc.
        indices = find (filtered_dat);
        [Y, X] = ind2sub(size(filtered_dat), indices);
        time_series_dat(i, 1) = mean(X);
        time_series_dat(i, 2) = mean(Y);

        % Play video if specified.
```

```
        if (play_vid)
            % Original video.
            subplot(2,2,1);
            imshow(uint8(frame_dat.cdata)); drawnow;

            % Black and white video.
            subplot(2,2,2);
            imshow(uint8(image_dat_bw)); drawnow;

            % Black and white video with crop.
            subplot(2,2,3);
            imshow(uint8(cropped_dat)); drawnow;

            % Black and white video with crop and filtered gray scale values.
            % filtered_dat consists of ones and zeros, where values greater
            % than gray_scale_val will be one & values less than or equal
            % to gray_scale_val will be zero. We multiply by 255 to amplify
            % these values and make them white.
            subplot(2,2,4);
            imshow(uint8(255 * filtered_dat)); drawnow;
            hold on;
            % Show time point used in PCA.
            plot(mean(X),mean(Y),'r*')
            hold off;
        end
    end

    if (play_vid)
        % Close video at end.
        close all;
    end
end
```

align_dat.m

```
%{
    Helper function where given positional data from three different
    cameras - crops each time series data as to be better aligned in time with
    eachother, wrt. to the oscillation they are all capturing.
    In addition - in order to perform PCA, each matrix must have the same
    dimensions - so we determine the matrix with the smallest number of
    frames and trim the other two to that size.
    Inputs are the three time series we wish to align in time and crop and
    the number of frames to find the lowest y-coordinate within in order to
    try and align the data within the same oscillation.
    Outputs all the time series combined in order to perform PCA.
%}

function [cropped_dat] = align_dat (time_series_dat_1, time_series_dat_2, time_series_dat_3,
num)
    [~, i] = min(time_series_dat_1(1:num, 2));
    time_series_dat_1  = time_series_dat_1(i:end, :);

    [~, i] = min(time_series_dat_2(1:num, 2));
    time_series_dat_2  = time_series_dat_2(i:end, :);

    [~, i] = min(time_series_dat_3(1:num, 2));
    time_series_dat_3  = time_series_dat_3(i:end, :);

    % Find minimum frame length and resize.
    min_frame = length(time_series_dat_1);
    if (length(time_series_dat_2) < min_frame)
        min_frame = length(time_series_dat_2);
    end
    if (length(time_series_dat_3) < min_frame)
        min_frame = length(time_series_dat_3);
    end

    % Set return value.
    cropped_dat = [time_series_dat_1(1:min_frame, :)'; ...
        time_series_dat_2(1:min_frame, :)'; time_series_dat_3(1:min_frame, :)'];
end
```