Assignment 3: Heisenberg's Uncertainty Principal

William Ojemann
AMATH 482
February 24th, 2021

Abstract

One of the many prevalent uses of digital signal processing and data analysis is in image compression, processing, and analysis. This report outlines the implantation of both image and data analysis methods to identify principles of motion in a mass spring system. Image thresholding and object extraction are used to identify the position of a mass on a spring and singular value decomposition (SVD) is used to extract the principal components of this motion in a series of four test cases. The outlined procedure gives a robust technical application of theoretical mathematics to a real-world system.

Introduction

The human visual and cognitive system is incredibly adept at both identifying moving objects in a vivid field of view as well as detecting patterns of motion. While we possess this innate skill, using computational methods to isolate objects in an image and extract the dynamics of motion poses a significantly greater challenge. There are many methods of altering an image for analysis ranging from simple thresholding to denoising by solving the partial differential equation for heat diffusion. In the context of this assignment, taking a video of a mass oscillating on a spring with varying types of initial conditions and movement and identifying the dynamics of the movement, isolating the object with simpler brightness thresholding methods is applicable in part because of the additional bright light on the moving mass.

While the image analysis component of the project is less complex, the bulk of the mathematical analysis lies in the implementation of SVD. SVD, developed more than a century ago independently by a handful of mathematicians[2], is a matrix decomposition that can be applied to most matrices. Foundational tools like the Fourier transform have given life to the field of signal analysis, but SVD has been the driving force behind modern computational analysis in engineering dynamics as well as signal and image processing. SVD is so important to data analysis and this project specifically because it breaks down a matrix into a series of orthogonal axes that can be used to analyze the important trends in a higher dimensional dataset.

To explore the applications of SVD, this report covers the decomposition of mass positions extracted from different videos in four different test cases: vertical movement, vertical movement with noise, vertical and horizontal movement, and vertical, horizontal, and rotational movement. With the object's position extracted from the videos, SVD is then employed to take in positions captured from three different camera angles to identify the principal components of motion. While a mass-spring system may seem relatively simple, the mathematics involved in analyzing the extracted object positions requires a background in SVD, proper orthogonal decomposition (POD), and principal component analysis (PCA) – which will be provided below.

Theoretical Background

POD and PCA are both powerful computational tools, but both depend on a background knowledge of SVD. SVD is based on the principle that for any matrix $A \in C^{m\,x\,n}$, there exists a singular value decomposition $A = U\Sigma V^*$ where * represents the complex conjugate transpose. The matrices that compose the SVD come from the foundational idea that the transformation A can be represented as a series of scaled unit vectors, u_n (1).

$Av_1 = \sigma_1 u_1$, $Av_2 = \sigma_2 u_2 \ldots Av_n = \sigma_n u_n$ eq. 1

$AV = U\Sigma$ eq. 2

In matrix form this equation takes on the form (2), where $V \in \mathbb{R}^{nxn}$ and its columns are dubbed the principal components, $U \in \mathbb{R}^{mxn}$, and Σ is a diagonal $\Sigma \in \mathbb{R}^{nxn}$ matrix, where U and V are unitary – meaning their columns form an orthonormal basis. The analytical solution to this equation is beyond the scope of this report, but it should be noted that this solution is technically the economical or reduced SVD, which is the form of the SVD used in the computations outlined in this procedure.

The two most important properties of the SVD in relation to this project are that the singular values that form the diagonal of the matrix Σ are non-negative, real, and sorted from largest to smallest. What this means is that the first singular value and its corresponding columns in U and V contain the most information about the matrix A of all the singular values. Going back to the vector representation of the decomposition of A, the equation $A = U\Sigma V^*$ can be represented by the sum of the individual vectors that make up U, V, and Σ (3).

$A = \sum_{k=1}^{r} \sigma_k u_k v_k^*$ eq. 3

This equation is so important because it demonstrates how the matrix A can be formed by a lower rank approximation using less of the singular values k and their corresponding orthogonal principal components, v_k, to reconstruct it (4).

$A_N = \sum_{k=1}^{N} \sigma_k u_k v_k^*$ eq. 4

Using the principle behind equation 4, PCA and POD both are methods of finding an orthonormal basis for the matrix A. Built upon the computation of the SVD, the components of the orthonormal bases are scaled by singular values and represent the axes along which the dataset varies the most in descending order. The main difference between PCA and POD for this assignment is that when performing PCA it is standard to demean the data and scale the variance of the raw dataset to one in order to standardize the effects of different data points and features on the decomposition.

Algorithm Implementation and Development

Analyzing and decomposing the movement of a mass on a spring from video requires two overarching steps: image processing, and signal decomposition. The image processing is achieved through a series of comprehensive user interfaces in the MATLAB command window. The first step in processing the images requires loading the video files into the MATLAB workspace using the eval command and watching each of the three camera angles for the four different test conditions. Iterating through each camera and test case, the first step in isolating the object is turning the frames of the videos from 3D color matrices to 2D gray scale images. With the gray scale images, narrowing the frame of analysis by collecting left, right, high, and low boundaries of object movement with the MATLAB 'getpts()' function allows for easier identification of the object using thresholding (**figure 0**). The x and y coordinates of the boundaries were stored as an offset for the object's position in each video. The grayscale cropped image is then binarized using logical indexing and a threshold that is iteratively changed by the user. In the thresholding process, if the thresholding process reveals the object's movement extends beyond the range of the cropped boundaries, the boundary collection process can be redone while still saving the image threshold.

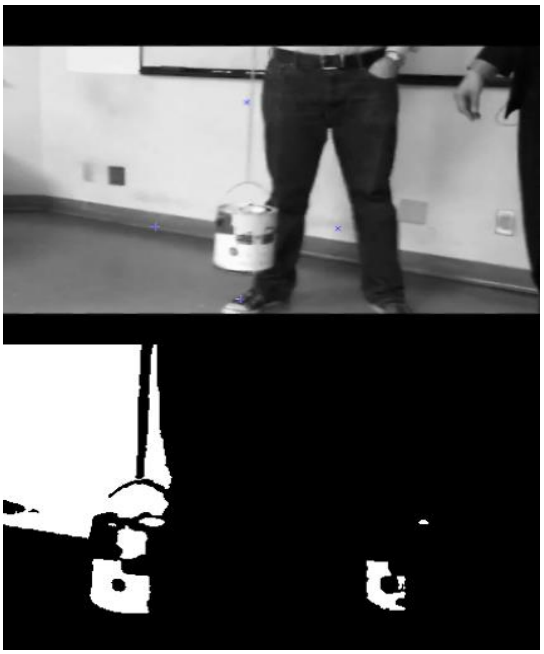


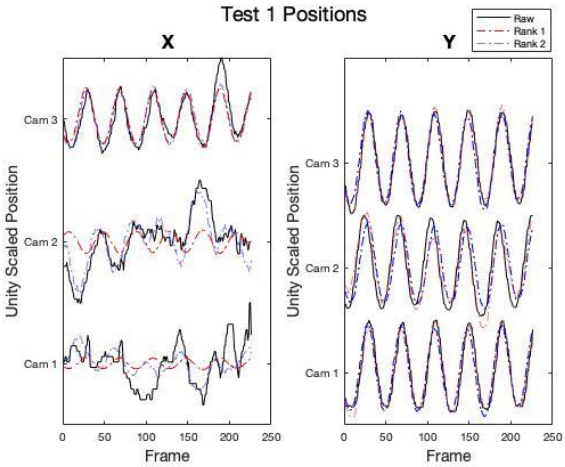
Figure 0: *(top) grayscale image of the object in motion with vertical and horizontal boundary points. (bot. left) initial binarization. (bot. right) Adjusted threshold for object isolation.*

With the image cropped and binarized, because this process did not yield exactly one pixel, the object's approximate position in each frame of each video for each test is calculated using the mean coordinate of the remaining true valued pixels. These approximate positions are concatenated into an array that contains the position of the object across time measured in frames. To avoid repeating work unnecessarily the data structures containing the thresholds, offsets, and object positions are saved to separate files in the current directory. Because the videos have varying lengths, using the internal 'resize_data' function, the object position arrays are cropped to the length of the shortest video. The resized arrays are then concatenated to create a matrix for each test condition that contains the x and y coordinates of the object position calculated from each of the three cameras. Because PCA requires the demeaning of data, the resizing function also subtracts the mean of each coordinate array. similarly, each matrix is scaled to an order of unity for simpler visualization and analysis and to account for the fact that the cameras were zoomed at different levels and camera 2 carried more weight than the other cameras. While this is technically neither PCA nor POD, it represented the best solution to this particular problem. Because some of the cameras started at different points in the object's motion, custom offsets were set for specific videos to properly align the data.

While the mathematical foundation of SVD, and POD is fairly complicated, because of how useful and prevalent the process is the SVD of each test condition is calculated simply using MATLAB's 'svd' function. Using one and two principal components lower rank reconstructions of the data are created and the reconstructions along with the singular values and component dynamics are plotted for analysis. The energy of each singular value is calculated using the L2 norm to elucidate the relative kinetic energy of each principal component relative to the total system.

In order to relate the SVD and POD back to the mass spring system, the code outlines a frequency analysis of the first and second

Figure 1: *Test 1 coordinates of object in (left) x and (right) y directions with rank 1 and 2 approximations.*



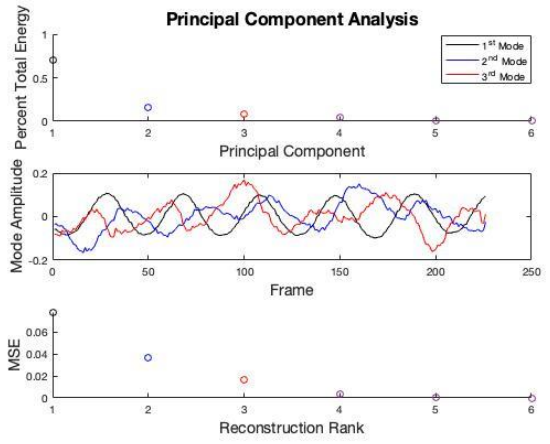


Figure 2: *Test 1 (top) percent system kinetic energy of each singular value, (mid) first three principal components, and (bot) mean squared error of different rank reconstruction.*

principal components of the third test system. This analysis is performed using the fast Fourier transform and finding the maximum frequency of the component's oscillation. This maximum frequency identifies the primary frequencies of the systems oscillation.

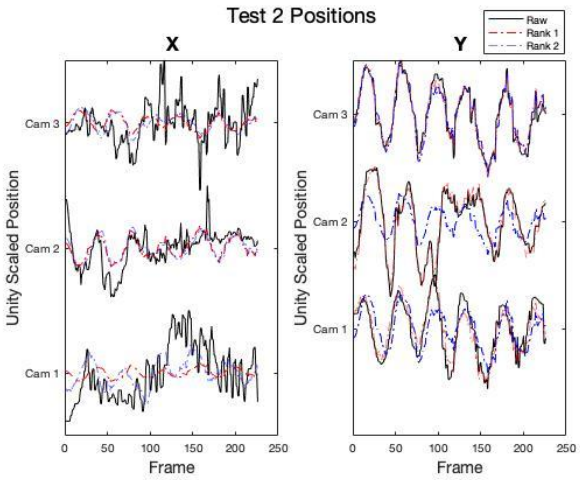
Computational Results

For the first test condition with simply vertical movement imposed on the mass and spring system the object's movement in each camera and the resulting decomposition were plotted. Plotting the extracted coordinates (**figure 1**) shows that the primary axis of movement is up and down along the y axis in accordance with the test parameters. While there was some oscillation in the x axis for test one, reconstruction with one and two principal components show that most of this variation is not contained in the first components but rather noise explained in later components. This is confirmed by looking at the energy of the singular values of the camera system (**figure 2, top**) where the first component contains almost all of the energy. The first

The principal components and object location for the second test condition containing primarily vertical mass movement with the addition of noise in the form of camera shape were also plotted for analysis. The presence of noise is extremely prevalent in the plot showing the raw coordinates of the object as well as the one and two rank reconstructions of the data (**figure 3**). The noise is also visible in the reconstruction error (**figure 4, bot.**) where the decrease in error was gradual as opposed to dropping off after one specific component. In this test the data from camera two's y movement could be reasonably reconstructed from the first two principal components, but for the most part the lower rank reconstructions were unable to match the raw data in both the x and y oscillations. The first three modes of the test two spring mass system show mostly noise (**figure 4, mid**). This is reiterated in the error of reconstruction (**figure 4, bot**), which shows higher error than the rank one reconstruction without noise.

The analysis of the conditions of test three consisted of similar plotting of the object's oscillation and the principal components of

Figure 3: *Test 1 coordinates of object in (left) x and (right) y directions with rank 1 and 2 approximations.*



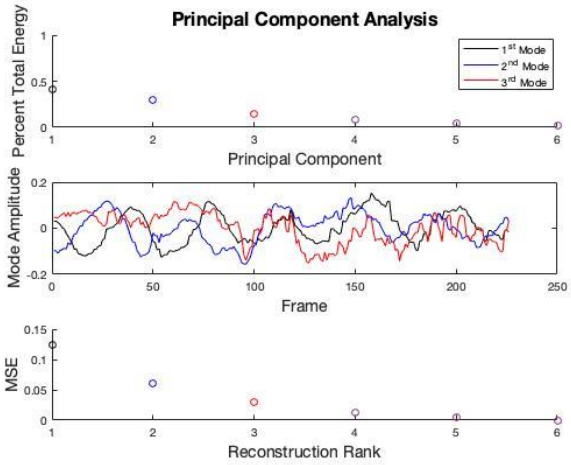
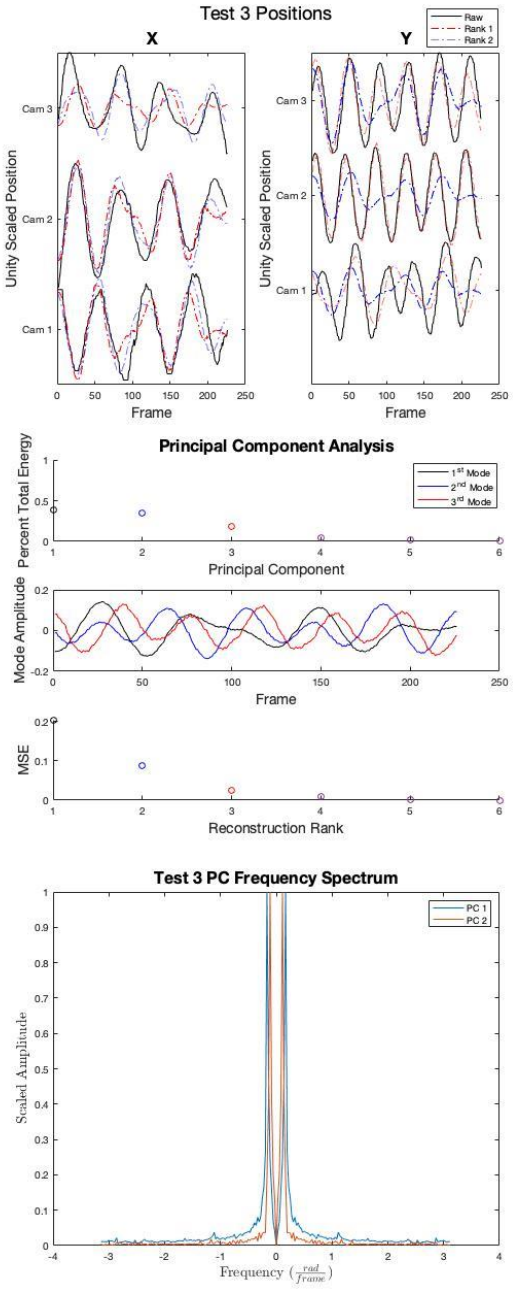


Figure 4: *Test 2 (top) percent system kinetic energy of each singular value, (mid) first three principal components, and (bot) mean squared error of different rank reconstruction.*

the object's dynamics. The raw data from all cameras in test three show strong oscillatory motion in the x and y directions (**figure 5a**) consistent with the 2D oscillation conditions of the test. The two-dimensionality of the data is represented in the low rank reconstructions where information added by the second principal component skews the data away from oscillation in the y direction and better explains oscillation in the x direction. Similarly, the reconstruction error is higher than any other test with just a rank one reconstruction as the test 3 system with oscillations in orthogonal directions cannot be fully explained by one dimension (**figure 5b, bot**). This trend is present in the principal components as well (**figure 5b, top**) where the first two principal components contain about equal amounts of the energy in the system. Exploring the test 3 case more, the frequency spectrum of the first two SVD modes shows that while they share similar envelopes, the peak frequency for two components is offset as they explain different oscillations (**figure 5c**).

In test 4 with the introduction of rotation into two-dimensional movement, the traces of the object coordinates as well as the results from

Figure 5: *(a) Test 3 coordinates of object in (left) x and (right) y directions with rank 1 and 2 approximations. (b) Test 3 (top) percent system kinetic energy of each singular value, (mid) first three principal components, and (bot) mean squared error of different rank reconstruction. (c) Frequency spectrum of first two principal components with maximum frequencies at .17 and .11 radians per frame respectively.*

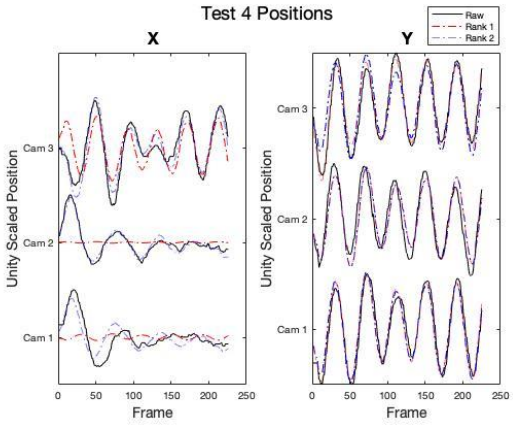


principal component analysis were plotted. While the system contained 3 separate types of motion, the results of image analysis and SVD were most similar to the plots of one-dimensional motion (**figure 6, figure 7**). This may be due to energy being transferred between the x direction of movement, which dampens, and the rotation, which was not picked up in the image processing procedure.

Conclusion & Summary

Through the exploration of a mass spring system this assignment demonstrated the power of SVD and POD or PCA. POD was able to identify the oscillatory nature of a mass spring system in multiple dimensions and to identify principal components that explained this variance. POD and PCA are powerful computational tools that can be applied well beyond this experiment to identify significant trends in extremely complex and multidimensional data.

Figure 6: *Test 4 coordinates of object in (left) x and (right) y directions with rank 1 and 2 approximations.*



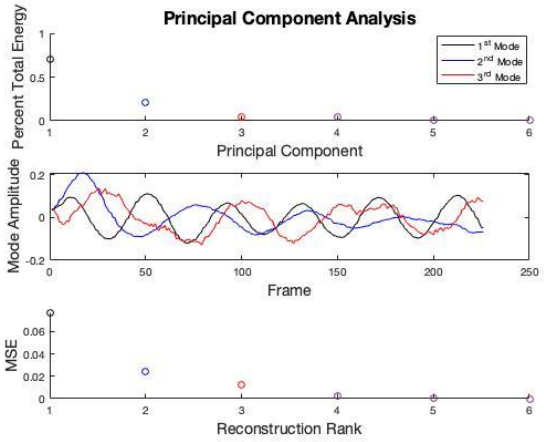


Figure 7: *Test 4 (top) percent system kinetic energy of each singular value, (mid) first three principal components, and (bot) mean squared error of different rank reconstruction.*

Appendix A – MATLAB Functions

x = cell(m,n) – Initializes an empty m x n cell, used to initialize all storage variables for the different tests and datum with varying sample numbers.

x = ones(m,n) – Initializes a m x n matrix of ones, used to initialize matrices.

x = zeros(m,n) – Initializes a m x n matrix of zeros, used to initialize matrices

x = eval('x') – runs the MATLAB code x, used to load the video files into a temporary variable

Xgray = rgb2gray(X) – converts an m x n x 3 color image to an m x n grayscale image.

ischar(x) – returns a Boolean 0 if x is not a char and a 1 if x is a char. Used for the user interface to determine user input type.

isnan(x) – returns a Boolean 0 if x is not a NaN and a 1 if x is a NaN. Used for the user interface to determine user input type.

check = input('input') – Takes in a user input to the prompt 'input' and returns it to check, used for the user interface.

[x,y] = find(X) – Takes a matrix X and returns the x and y indices of any non-zero coordinates, used to locate the object within the video frame.

plot(t,y) – plots the signal y against its time domain t, used to plot the location of the object using frames as a unit for time.

scatter(t,k) – plots the individual data points as dots, used to plot the singular values, their energies, and the reconstruction

round(x) – rounds the value x to the nearest integer, used to ensure values used as indices were integers.

linspace(start, end, int) – creates an array of variables between the start and end value with int data points, used to create axis ticks for plotting.

sum(x) – returns the sum of the values in x with the option to sum along one or more dimensions, used to calculate errors.

[U,S,V] = sum(x, 'econ') – returns the economical singular value decomposition of the matrix x.

svd_analysis_plotting(data,testset,fignum,camcount) – Internal function used to calculate the SVD for each test and plot the figures used for analysis of the principal components and system as a whole.

[unity_data,M,x_coordinates_crop, y_coordinates_crop] =
resize_data(x_coord,y_coord,tests,camcount) – function used to crop and concatenate data from different cameras for each test and return the data demeaned for SVD and put onto order of unity for plotting.

[coordinates, thresholds, click_coordinates] = load_data() – Internal function used to load previously acquired object positions, thresholds, and boundary coordinates.

parametrize(camcount, tests) – Internal function used to create cells containing the object names for each recording.

plot_pc_analysis(data) – Internal function used to plot the spectral analysis of the first two principal components of test 3.

Appendix B – MATLAB Code

```
% William Ojemann
% AMATH 482
% 16 Feb 2021
% Assignment 3
clear; close all; clc;
%% Data Parameters
camcount = 3;
tests = 4;
[cams, vids] = parametrize(camcount,tests);
%% Data Aquisition
% clc
% xclick_store = cell(tests,camcount);
% yclick_store = cell(tests,camcount);
% thresh = ones(tests,camcount)*220;
% thresh_check = ones(tests,camcount);
% x_coordinates = cell(tests,camcount);
% y_coordinates = cell(tests,camcount);
% for test = 1:tests
% for cam = 2:camcount
%     xs = []; ys = [];
%     load(cams{test,cam});
%     vid_iter_temp = eval(vids{test,cam});
%     numFrames = size(vid_iter_temp,4);
%     clear vid_iter_gray;
%     for k = 1:numFrames
%         if cam == 3
%             vid_iter_gray(:,:,k) = rgb2gray(vid_iter_temp(:,:,:,k))';
%         else
%             vid_iter_gray(:,:,k) = rgb2gray(vid_iter_temp(:,:,:,k));
%         end
%     end
%     [offx,boundx,offy,boundy] = loc(vid_iter_gray,cams,test,cam,100);
%     xclick_store{test,cam} = [offx boundx];
%     yclick_store{test,cam} = [offy boundy];
%
%     check = 0;
%     while ~ischar(check) && thresh_check(test,cam)
%         for i = 1:4:numFrames
%             X_window = vid_iter_gray(offy:boundy,offx:boundx,i);
%             figure(101)
%             imshow(X_window>=thresh(test,cam));drawnow
%         end
%         disp(['Threshold: ' num2str(thresh(test,cam))]);
%         check = input('Give thresh. diff. (e.g. 10,-5), "reloc" to change
bounds, and y when done: ','s');
%         if check == "reloc"
%             [offx,boundx,offy,boundy] =
loc(vid_iter_gray,cams,test,cam,100);
%             check = 0;
%             continue
%         end
%         check = str2double(check); % returns Nan if not a number char
%         if ~isnan(check) % checks for number
%             thresh(test,cam) = thresh(test,cam) + check; %adds number to
thresh
```

```
%         else
%             check = 'done'; % ends loop if non number was input
%         end
%     end
%
%     for j = 1:numFrames
%         X_window = vid_iter_gray(offy:boundy,offx:boundx,j);
%         X_binary = X_window >= thresh(test,cam);
%         if sum(sum(X_binary))
%             [y,x] = find(X_binary); %(:,200:end));
%             x = round(mean(x));
%             y = round(mean(y));
%             Y = zeros(size(X_window)); %(:,200:end)));
%             Y(x,y) = 1;
%             xs = [xs x+offx];
%             ys = [ys y+offy];
%         else
%             xs = [xs xs(end)];
%             ys = [ys ys(end)];
%         end
%     end
%     x_coordinates{test,cam} = xs;
%     y_coordinates{test,cam} = ys;
% end
% end
% for i = 1:tests
%     figure(i)
%     subplot(2,1,1)
%     plot(x_coordinates{i,1})
%     subplot(2,1,2)
%     plot(y_coordinates{i,1})
% end

%% File Storage
% % clicks.x = xclick_store;
% % clicks.y = yclick_store;
% % save('clicks.mat', '-struct', 'clicks');
% % coordinates.x = x_coordinates;
% % coordinates.y = y_coordinates;
% % save('coordinates.mat', '-struct', 'coordinates');
% % save('thresholds.mat','thresh');
%% Load Files
[coordinates, thresholds, click_coordinates] = load_data();
x_coordinates = coordinates.x;
y_coordinates = coordinates.y;
%% Data formatting
[data,M,~,~] = resize_data(x_coordinates,y_coordinates,tests,camcount);
%% Principal Component Analysis
for i = 1:tests
    svd_analysis_plotting(data,i,i*10, camcount)
end
%% Analysis of Principal Components
plot_pc_analysis(data);
%% Functions
function svd_analysis_plotting(data,testset,fignum,camcount)
    %%%%% Plotting Reconstructions
    % Perform SVD
```

```
    [U,S,V] = svd(data{testset},'econ');
    % Create rank 1 and 2 reconstructions
    low_rank_one = U(:,1)*S(1,1)*V(:,1)';
    low_rank_two = U(:,1:2)*S(1:2,1:2)*V(:,1:2)';
    figure(fignum)
    % Iterate through feature sets and plot approximations along with raw
    % data for each X and Y directions.
    for i = 1:camcount
        l = length(data{testset});
        subplot(1,2,1)
        plot(1:l,data{testset}(2*i-1,:)+2*i,'k-')
        hold on
        plot(1:l,low_rank_one(2*i-1,:)+2*i,'r-.')
        p1 = plot(1:l,low_rank_two(2*i-1,:)+2*i,'b-.');
        p1.Color(4) = 0.5;
        legend('Raw','Rank 1','Rank 2')
        yticks(linspace(1,camcount,camcount)*2)
        yticklabels({'Cam 1', 'Cam 2', 'Cam 3'})
        title('X','Fontsize',16);
        ylabel('Unity Scaled Position','Fontsize',14)
        xlabel('Frame','Fontsize',14)
        subplot(1,2,2)
        plot(1:l,data{testset}(2*i,:)+2*i,'k-')
        hold on
        plot(1:l,low_rank_one(2*i,:)+2*i,'b-.')
        p2 = plot(1:l,low_rank_two(2*i,:)+2*i,'r-.');
        p2.Color(4) = 0.5;
        yticks(linspace(1,camcount,camcount)*2)
        yticklabels({'Cam 1', 'Cam 2', 'Cam 3'})
        title('Y','Fontsize',16);
        ylabel('Unity Scaled Position','Fontsize',14)
        xlabel('Frame','Fontsize',14)
    end
    hold off
    sgtitle(['Test ', num2str(testset), ' Positions'],'Fontsize',18);

    %%%%% Plotting Decomposition Properties
    figure(fignum+1)
    % Plot the percent total kinetic energy
    subplot(3,1,1)
    sigmas = diag(S);
    energies = zeros(1,length(sigmas));
    errors = zeros(1,length(sigmas));
    for i = 1:length(sigmas)
        energies(i) = sum(sigmas(i).^2)/sum(sigmas.^2);
        low_rank_recon = U(:,1:i)*S(1:i,1:i)*V(:,1:i)';
        errors(i) = sum((data{testset} -
low_rank_recon).^2,'all')./numel(low_rank_recon);
    end
    scatter(1,energies(1),'k');
    hold on
    scatter(2,energies(2),'b')
    scatter(3,energies(3),'r')
    scatter(4:length(sigmas),energies(4:end))
    hold off
    ylim([0 1])
    ylabel('Percent Total Energy','Fontsize',14);
```

```
    xlabel('Principal Component','Fontsize',14)
    title('Principal Component Analysis','Fontsize',16)
    xticks(1:length(sigmas))
    % Plot the MSE of reconstruction
    subplot(3,1,3)
    scatter(1,errors(1),'k');
    hold on
    scatter(2,errors(2),'b');
    scatter(3,errors(3),'r');
    scatter(4:length(sigmas),errors(4:end));
    hold off
    ylabel('MSE','Fontsize',14);
    xlabel('Reconstruction Rank','Fontsize',14);
    xticks(1:length(sigmas))
    % Plot the time evolution of each principal component
    subplot(3,1,2)
    plot(1:l,V(:,1)','k',1:l,V(:,2)','b',1:l,V(:,3)','r')
    ylabel('Mode Amplitude','Fontsize',14)
    xlabel('Frame','Fontsize',14)
    legend('1^{st} Mode', '2^{nd} Mode', '3^{rd} Mode')
end

function [unity_data,M,x_coordinates_crop, y_coordinates_crop] =
resize_data(x_coordinates,y_coordinates,tests,camcount)
    % Initialize cropped data
    x_coordinates_crop = cell(tests,camcount);
    y_coordinates_crop = cell(tests,camcount);
    data = cell(1,tests);
    longest_film = ones(1,tests)*10000;
    demeaned_data = cell(1,tests);
    unity_data = cell(1,tests);

    % Iterate through test
    for j = 1:tests
        % Find the shortest camera frame count for
        % each test
        for i = 1:camcount
            if length(x_coordinates{j,i}) < longest_film(j)
                longest_film(j) = length(x_coordinates{i});
            end
        end
        % Resize the data into the cropped format
        data{j} = nan(2*camcount,longest_film(j));

        for i = 1:camcount
            if i == 2 && j~= 3 && j~= 4
                x_coordinates_crop{j,i} =
x_coordinates{j,i}(16:longest_film(j)+15);
                y_coordinates_crop{j,i} =
y_coordinates{j,i}(16:longest_film(j)+15);
            elseif j == 4 && i == 2
                x_coordinates_crop{j,i} =
x_coordinates{j,i}(11:longest_film(j)+10);
                y_coordinates_crop{j,i} =
y_coordinates{j,i}(11:longest_film(j)+10);
            else
```

```
            x_coordinates_crop{j,i} = x_coordinates{j,i}(1:longest_film(j));
            y_coordinates_crop{j,i} = y_coordinates{j,i}(1:longest_film(j));
            end
            data{j}(2*i-1:2*i,:) = [x_coordinates_crop{j,i} ;
y_coordinates_crop{j,i}];
        end
        % Demean data for PCA and find maximum value for unity scaling
        demeaned_data{j} = data{j} - mean(data{j},2);
        M = max(demeaned_data{j},[],2);
        unity_data{j} = demeaned_data{j} ./ M;
    end
end

function [offx,boundx,offy,boundy] = loc(image,cams,test,cam,fignum)
    figure(fignum)
    for j = 1:4:size(image,3)
        imshow(image(:,:,j)); drawnow
        title(cams{test,cam})
    end
    disp('Pick two points as left & right bounds of movement');
    [xclick,~] = getpts;
    disp('Now pick low and high bounds');
    [~,yclick] = getpts;
    offx = round(xclick(1));
    boundx = round(xclick(2));
    offy = round(yclick(2));
    boundy = round(yclick(1));
end

function [coordinates, thresholds, click_coordinates] = load_data()
    % Internal Function
    % Load previously acquired data
    coordinates = load('coordinates.mat');
    thresholds = load('coordinates.mat');
    click_coordinates = load('coordinates.mat');
end

function [cams, vids] = parametrize(camcount,tests)
    % Internal Function
    % Initialize variables for loading data
    cams = cell(tests,camcount);
    vids = cell(tests,camcount);
    for cam = 1:camcount
        for test = 1:tests
            cams{test,cam} = ['cam' num2str(cam) '_' num2str(test) '.mat'];
            vids{test,cam} = ['vidFrames' num2str(cam) '_' num2str(test)];
        end
    end
end

function plot_pc_analysis(data)
figure
%for i = 1:4
    [~,~,v] = svd(data{3},'econ');
    %subplot(2,2,i)
    n = length(v);
```

```
    k = 2*pi/n*[0:(n/2 - 1) -n/2:-1]; ks = fftshift(k);
    f1 = abs(fftshift(fft(v(:,1))));
    f2 = abs(fftshift(fft(v(:,2))));
    [fmax,fmaxidx] = max(f1);
    [fmax2,fmaxidx2] = max(f2);
    plot(ks,f1/fmax);
    hold on
    plot(ks,f2/fmax2);
    title(['Test ',num2str(3),' PC Frequency Spectrum'],'Fontsize',16);
    ylabel('Scaled Amplitude','Interpreter','latex','Fontsize',14)
    xlabel('Frequency
($\frac{rad}{frame}$)','Interpreter','latex','Fontsize',14)
    legend('PC 1', 'PC 2');
    disp(num2str(abs(ks(fmaxidx))));
    disp(num2str(abs(ks(fmaxidx2))));
%end
end
```