AMATH 482 Homework 3

Sophia Jannetty

February 24, 2021

Abstract

Three cameras recorded a paint can with a pink flashlight attached to its top bouncing up and down. Here I take these videos as input, extract the location of the can in each video frame, and use Principal Component Analysis (PCA) to ascertain information about the paint can's movement. I attempt this analysis under four conditions: an ideal case (case 1) in which the cameras are steady and the can bounces only up and down, a noisy case (case 2) in which the cameras shake and the can bounces only up and down, a horizontal displacement case (case 3) in which the can is bouncing up and down while swinging side to side, and a horizontal displacement and rotation case (case 4) in which the can is bouncing up and down, swinging side to side, and rotating about the string from which it hangs. The PCA was most effective at identifying the motion in cases 1 and 3 and was least effective in cases 2 and 4.

1 Introduction and Overview

Principal Component Analysis (PCA) involves performing the Singular Value Decomposition (SVD) on a pre-processed data matrix. The SVD allows for the identification of the primary directions of variance present in a dataset, enabling the production of low-dimensional reductions of high-dimensional data that preserve the dynamics and behavior of the system [2]. In this exercise, three cameras are recording the motion of a paint can. An X and Y value for the location of the can was identified for each frame of each camera's recording. This resulted in a matrix containing six dimensions of input data for each case. This matrix was pre-processed and run through the SVD, resulting in a decomposition of the matrix into constituent matrices that describe the basis vectors of the data and the relative magnitudes of those basis vectors in the data. The resulting basis vectors and singular values were inspected to see whether PCA was able to accurately capture the motion in the system in each condition. It was most successful in cases 1 and 3 and least successful in cases 2 and 4.

2 Theoretical Background

2.1 Singular Value Decomposition (SVD)

2.1.1 Overview and Definition of Components

Assume we have a large data set $X \in \mathbb{C}^{nxm}$ in which each column x_k contains data from a distinct instance of measurement (for example, each column contains the x or y location of the can found from analyzing one camera):

$$X = \begin{bmatrix} | & | & & | \\ a_1 & a_2 & \dots & a_m \\ | & | & & | \end{bmatrix} \tag{1}$$

The SVD expresses this matrix as the product of three constituent matrices as shown in equation 2.

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

In this equation, U is a nxn unitary matrix with orthonormal columns, V is an mxm unitary matrix with orthonormal columns, and Σ is an mxm diagonal matrix thats number of nonzero elements is equal to the rank of X. The nonzero elements of Σ will be, at most, the smaller dimension of matrix X (n or m). The economy SVD is a version of the SVD that minimizes the dimensions of these matrices to eliminate the zero

entries on Σ (this reduced Σ is now called $\hat{\Sigma}$) and to eliminate the corresponding columns of U (this reduced U is now called $\hat{U}$. Equation 3 shows to reduced SVD [1]. I used the reduced SVD for this project so for the remainder of this document I will use the terminology associated with the reduced SVD (though all concepts are the same for the full SVD).

$$X = \hat{U}\hat{\Sigma}V^* \tag{3}$$

The values σ_n along the diagonal of $\hat{\Sigma}$ are called the singular values of matrix A and are ordered from smallest to largest. The vectors u_n that make up the columns of $\hat{U}$ are called the left singular values of X and the vectors v_n that make up the columns of V are called the right singular values of X (note that the MATLAB implementation of the econ SVD returns matrices $\hat{U}$, $\hat{\Sigma}$, and V, NOT the complex conjugate transpose V^* in Equation 3. In this document I will describe V, not V^*). This left/right terminology relates to the geometric understanding of what each element of the SVD is doing and the resulting derivation [3]. These details are beyond the scope of this paper. Instead I will present a conceptual overview of the contents of each matrix.

The columns of $\hat{U}$ will have the same dimensions as the columns of X. Each column contains a basis vector that describes some amount of variance present in the columns of X. These basis vectors are orthonormal and are hierarchically arranged such that the basis vector in the first column describes more of the variance in the columns of X than the basis vector in the second column, and so on. The number of columns will be equal to the number of nonzero entries in $\hat{\Sigma}$.

The singular values held in $\hat{Sigma}$ are all non-negative and represent the energy in the dataset captured by the corresponding basis vector, or the relative amount of variance in the cols of X that is in the direction of the corresponding basis vector. These values are also hierarchically arranged, such that the first value is the largest and the value in each subsequent row/col is less than the value in the previous row/col [2]. The value in each row/col m corresponds to the basis vector in $\hat{U}[:, m]$.

Each columns of V also corresponds to a basis vector (the m column corresponds to the basis vector in the m col of $\hat{U}$, however each row n corresponds to a the column n of X. Each element [n,m] is the projection of the mth standard basis vector onto the nth column of X. Conceptually, this tells me how much of the nth column of X is in the direction of the mth standard basis vector. If talking about the conjugate transpose of V V^*, each entry [n,m] of V^* tells gives the inner product between the nth column of X and the mth column of $\hat{U}$, or the mth basis vector.

2.1.2 Use for Matrix Approximation

The SVD provides a hierarchy of optimal low-rank approximation of matrix X. Matrix X is the sum of r rank-one matrices calculated in Equation 4 [2].

$$X = \sum_{j=1}^{r} \sigma_j u_j v_j^* \tag{4}$$

The SVD can be used to see how many independent basis vectors are present in the dataset and will order them from most to least important. By determining the proportion of the overall energy in the matrix each basis vector describes (found by dividing each singular value by the sum of all the singular values) you can set a threshold for how much of the energy you want represented in your matrix approximation and determine how many singular values (or ranks) you need to keep to reach that threshold. By setting r to this number of ranks in Equation 4, you can create a low-rank approximation of your input Matrix and can eliminate all information related to extraneous singular values. This allows for accurate and efficient matrix reduction.

2.2 Principal Component Analysis (PCA)

PCA requires pre-processing the input data before performing SVD. PCA literature depicts the example matrix in the opposite orientation compared to SVD literature, but I will keep with same orientation as described above. The mean of each column k of X is subtracted from each element in column k such that the mean of each column is set to zero. Additionally, every element in the matrix is divided by $\sqrt{n-1}$. This sets the variance of the data to one. The PCA is then performed on this pre-processing data. Because of the pre-processing, the resultant coordinate system will be defined by principal components that are uncorrelated to each other (orthogonal) but that have maximal correlation with the measurements in X [1].

3 Algorithm Implementation and Development

3.1 Can Location Extraction

The first frame of each video was inspected to identify the approximate initial location of the light on the top of the can. An appropriate search area size was specified for each video condition and for the first frame this area was centered around the approximate location of the light found from initial visual inspection. The window was made to be larger in video clips where the can moved a large distance between frames and smaller when the can didn't move very far between frames. For each frame, this window was searched for either a white or a pink pixel (depending on the case and whether or not the bright point of the light was always visible to the camera, I searched for a near-white pixel in every video except case 3 camera 1, case 3 camera 2, case 3 camera 3, case 4 camera 1 and case 4 camera 2). I implemented this search by searching for pixels that had r, g, and b values within certain range. For white all three had to be over either 245, 240, or 235 depending on the camera and instance. For pink the r value had to be between 240 and 255, the g value had to be between 19 and 200, and the b value had to be between 130 and 205.

The first pixel found in the window that was within the color thresholds signified (so that was either sufficiently white or sufficiently pink) was recorded as the location of the can in that frame. The x location was saved to an x location vector and the y location was saved to a y location vector. The search window was then centered on this point and this relocated search window was the area searched for a white or pink pixel in the next frame. If a white or pink pixel was not found in the search window of a frame, the previous frame's can location was recorded as the current frame's can location and the search window was not moved.

3.2 Video Synchronizing

Following the can location extraction process, the can location data was held in six vectors, one x location vector and one y location vector for each camera. These data were plotted to determine the offset in timing between each camera. The frame that held the first identifiable peak or trough of the can's bounce was identified for each camera. Frames were then trimmed off the beginning of two of the videos and the can location extraction was repeated so that the first peak or trough of the can's motion occurred on the same frame for all cameras.

3.3 Data Pre-Processing and SVD

The six location vectors were trimmed to 200 frames to eliminate some bad tracking artifacts that occurred near the end of the videos. They were then combined into a 6x200 matrix such that each row of the matrix was a different vector. The mean of each row was subtracted from each element of each row such that the mean of each row was made to be zero. The matrix was then transposed, divided by $\sqrt{(number of elements) - 1}$, and fed into MATLAB's svd function.

3.4 Plotting Results

For each condition I plotted each basis vector (each column of $\hat{U}$ scaled to its principal component), the original data matrix, the rank one approximation of the matrix and the minimal-rank approximation of the data required to capture 80% of the energy in the system. This 80% threshold was determined by visually inspecting the matrix approximations capturing %70, 80%, and 90% of the motion in the system for each case and deciding that 80 looked close enough. I wanted to use the minimal percent that still looked reasonable in order to maximize the potential for data compression.

To find the number of ranks needed to surpass this 80% threshold I first divided each sigma value by the sum of all the sigma values to determine the percent of energy in the system described by each basis vector. I looked at the percent energy associated with the first principal component to see if it was over .8. If not, I added the percent energy associated with the next principal component and asked if that sum was over .8. I continued this until I broke the 80% threshold. The number of principal component energies I needed to break this threshold determined the number of ranks I needed to incorporate in order to represent 80% of the energy in the system. I calculated this rank's approximation for each condition using Equation 4.

4 Computational Results

Full plots of the input data, the 1-rank approximation, the minimal rank approximation needed to capture 80% of the energy in the system, the scaled basis vectors, and the unscaled basis vectors can be found in

Appendix 3. Table 1 in Appendix 3 contains, for each case, the values of the principal components needed to capture 80% of the energy in the input matrix. All plots for cases 3 and 4 are in Appendix 3.

4.1 Case 1: Ideal

The raw data from the can tracking in case 1 can be seen in Figure 1 a. Tracking for this case was largely successful. The shape of the basis vector with the largest singular value is sinusoidal (see Figure 1 d.), suggesting that the vertical bouncing of the can was the primary motion detected in the matrix. Two ranks are required to capture 80% of the motion in matrix X (see rank 2 approximation in Figure 1 b. and principal component values in Table 1 in Appendix 3). Though the second basis vector is required to capture 80% of the energy in matrix X, I am not sure how to interpret what motion the second basis vector is describing. If my tracking and my synchronizing of the cameras had been perfect I would have expected to see all of the motion in one principal component. On the whole PCA effectively identified the dominant motion in this dataset.

4.2 Case 2; Noisy

The raw data from the can tracking in case 1 can be seen in Figure 2 a. Tracking for this case was not nearly as successful as Case 1; the raw data are not very smooth. This is reflected in the shapes of the basis vector with large singular values (seen in Figure 2 d-g), suggesting that the PCA analysis did not interpret the primary motion in the matrix to be sinusoidal. This is reasonable given that my input data was so scattered. Four ranks are required to capture 80% of the motion in matrix X (see Figure 2 c, principal component values can be found in Table 1 in Appendix 3). I do not know specifically what motion to attribute the second, third, and fourth basis vectors to, but because my data were so noisy it is reasonable that there was detectable motion in many directions as the recorded location of the can jumped around between frames.

4.3 Case 3 : Horizontal Displacement

The raw data from the can tracking in case 1 can be seen in Figure 7 in Appendix C. Tracking for this case was fairly successful; the raw data are quite smooth and both X and Y location vectors exhibit sinusoidal motion. This is reflected in the shapes of the basis vectors with large singular values (seen in Figure 8 in Appendix C). I think the first basis vector describes the motion in the up-and-down direction and the second basis vector describes the motion in the horizontal direction, suggesting that the PCA analysis interpreted the primary motion in both Cartesian directions to be sinusoidal. Four ranks are required to capture 80% of the motion in matrix X (see Figure 7 in Appendix C, principal component values can be found in Table 1 in Appendix 3). It is expected that more ranks would be required in this matrix than in the ideal case because there is motion in more than one direction. The third and fourth basis vectors are also lightly sinusoidal, and I am similarly not sure what motion they describe.

4.4 Case 4 : Horizontal Displacement and Rotation

The raw data from the can tracking in case 1 can be seen in Figure 9 in Appendix C. Tracking for this case was moderately successful at capturing the bouncing motion of the can but it is hard for me to tell whether my tracking function effectively tracked the rotation. Visually the raw data are generally smooth and both X and Y location vectors exhibit sinusoidal motion. This is reflected in the shape of the basis vector with the largest singular value (seen in Figure 10 in Appendix C). Four ranks are required to capture 80% of the motion in matrix X (see Figure 9 in Appendix C, principal component values can be found in Table 1 in Appendix 3). As in Case 3 is expected that more ranks would be required in this matrix than in the ideal case because there is motion in more than one direction. I think the first basis vector describes the motion in the up-and-down direction, but I am not sure what motion is described by the subsequent three basis vectors.

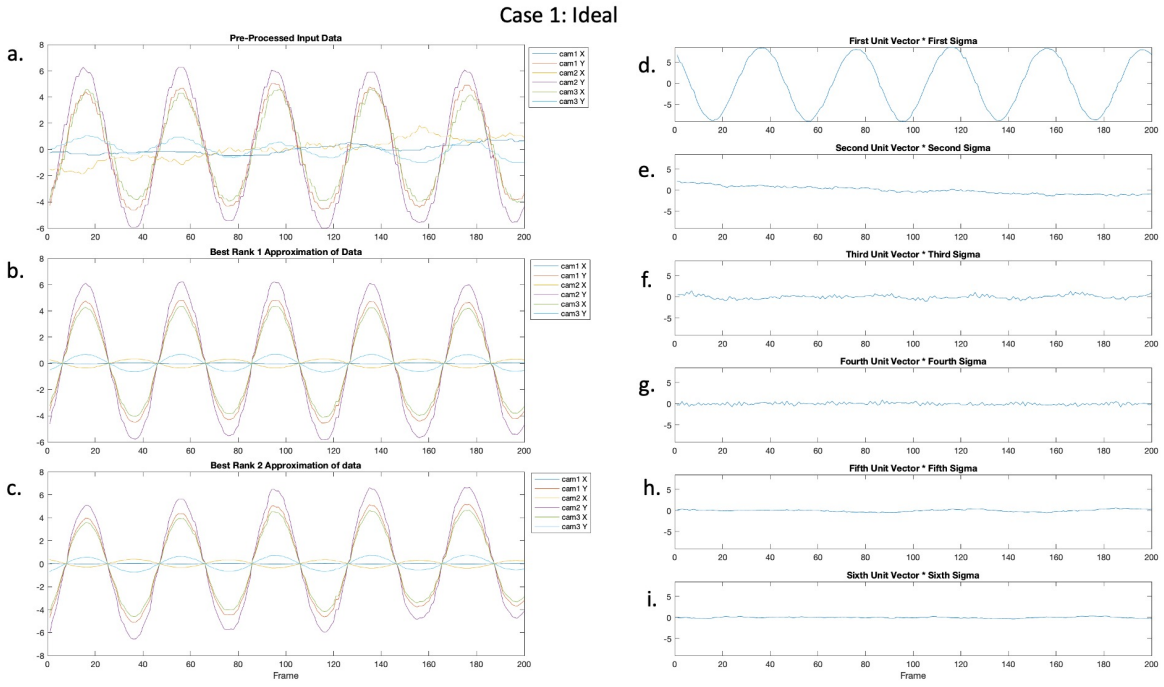

Figure 1: Ideal Case: a. Tracking data X after mean subtraction and setting variance to unity. b. and c. are rank 1 and rank 2 approximations of X respectively. d. - i. are basis vectors in order from greatest importance (d.) to least importance (i.) scaled to their respective singular values.

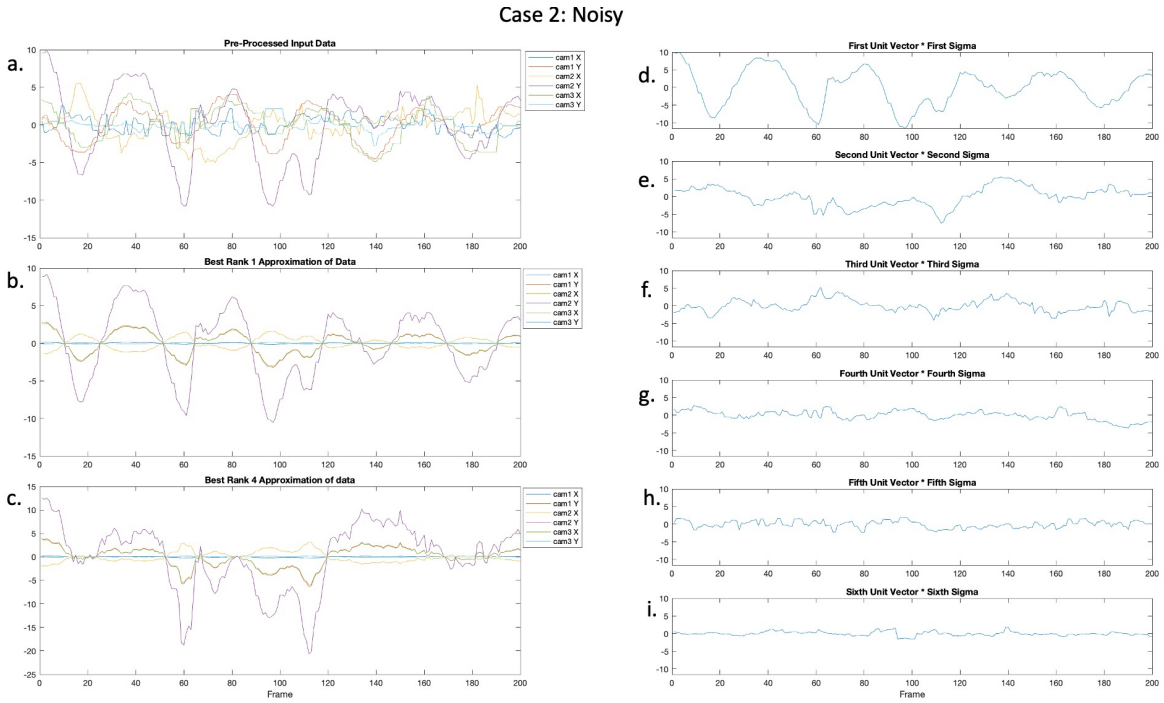

Figure 2: Noisy Case: a. Tracking data X after mean subtraction and setting variance to unity. b. and c. are rank 1 and rank 4 approximations of X respectively. d. - i. are basis vectors in order from greatest importance (d.) to least importance (i.) scaled to their respective singular values.

5 Summary and Conclusions

The two conditions in which my can location extraction was most accurate were cases 1 and 3. In these cases, the PCA identified principal components with sinusoidal shapes. Despite not knowing information about the equations governing the motion of the system, it was able to use the data to extract basis vectors that correctly describe the shape of the motions of the can.

My location extraction for case 2 was quite poor, and as a result the directions of motion extracted by the PCA do not reflect the motion of the can as accurately as was seen in case 1. This is expected as the SVD is determining the directions that contain the most variance in pixel values in my matrix X. By introducing noise variance was introduced to the data set and the SVD has no way of separating that variance from the variance due to the movement of the can.

My location extraction in case 4 was ok, but case 4 highlights a weakness of the SVD. Even if I had captured the rotation of the can perfectly, the PCA would produce two orthogonal components to describe the circular motion (one for the variance due to the rotation in the x direction and one for the variance due to the rotation in the y direction). It is possible that the waves seen in the third and fourth basis vectors from the case 4 analysis are describing the rotation of the can (see Figure 10 in Appendix C).

In addition to the rotation weakness outlined above, another weakness of the PCA in this context is that it required me to synchronize my videos. If I had not synchronized my videos, the PCA would not have been able to recognize the time delay and the motion of the can in one video clip would be interpreted as a motion independent to the motion of the can in another video clip. This would result in the motion of the can being described by multiple different basis vectors and principal components. Additionally, without knowing what motion you expect to see, it can be difficult to ascribe the principal components to specific features of the input dataset.

References

[1] Steven L. Brunton and Jose Nathan Kutz. *Data-driven science and Engineering: machine learning, dynamical systems, and control*. Cambridge University Press, 2019.

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

[3] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. SIAM, 1997. ISBN: 0898713617.

Appendix A MATLAB Functions

The first function is a MATLAB function. The other three are functions I wrote.

- `[uhat,shat,v] = svd(X'/sqrt(n-1), 'econ');` - produces an economy-size decomposition of m-by-n matrix A. If m ¿ n — Only the first n columns of U are computed, and S is n-by-n. If m = n — svd(A,'econ') is equivalent to svd(A). If m ¡ n — Only the first m columns of V are computed, and S is m-by-m.

- `[pixel_color_in_range] = in_range(pixr, pixg, pixb, rmin, rmax, gmin, gmax, bmin, bmax);` - takes a pixel and returns a boolean. True if rgb values of pixel are within parameter thresholds, otherwise false.

- `[normalized_energy] = find_svd(X)` - performs SVD, plots all my plots, returns vector of normalized principal components.

- `[] = plot_raw_data(xvec, yvec)` - plots my raw input data

- `[xloc, yloc] = track_light(vidframes, height, width, num_frames, rowmin, rowmax, colmin, colmax, rmin, rmax, gmin, gmax, bmin, bmax, window_side_length, parameter_optimize_mode)` - takes a video clip and tracks the light (by locating either a pink or white pixel in a search window of specified dimensions) in each frame. Returns vector of x and y locations of light for each video.

Appendix B MATLAB Code

```
%% Ideal Case
clear all; close all;
%% Camera 1
load cam1_1.mat;
[height1_1 width1_1 rgb1_1 num_frames1_1] = size(vidFrames1_1);
% align timing with falling of the can
vidFrames1_1=vidFrames1_1(:,:,:,15:num_frames1_1);
[height1_1 width1_1 rgb1_1 num_frames1_1] = size(vidFrames1_1);
%search from half way through cols to 2/3 of wat through cols
colmin = floor(width1_1/2);
colmax = (width1_1 - floor(width1_1/3));
%look for white pixel
min_color_val = 245;
max_color_val = 255;
window_side_length = 20;
rowmin = 1;
rowmax = height1_1;
[x1loc, y1loc] = track_light(vidFrames1_1, height1_1, width1_1,...
    num_frames1_1, rowmin, rowmax, colmin, colmax, ...
    min_color_val, max_color_val, min_color_val, max_color_val,...
    min_color_val, max_color_val, window_side_length, false);

%plot_raw_data(x1loc, y1loc);

%% Camera 2
load cam2_1.mat;
[height2_1 width2_1 rgb2_1 num_frames2_1] = size(vidFrames2_1);
% align timing with falling of the can
vidFrames2_1=vidFrames2_1(:,:,:,25:num_frames2_1);
[height2_1 width2_1 rgb2_1 num_frames2_1] = size(vidFrames2_1);

rowmin2 = 1;
rowmax2 = height2_1;
%search only middle third of columns
colmin2 = floor(width2_1/3);
colmax2 = width2_1 - floor(width2_1/3);
%look for white pixel
min_color_val = 245;
max_color_val = 255;
window_side_length = 20;
[x2loc, y2loc] = track_light(vidFrames2_1, height2_1, width2_1,...
    num_frames2_1, rowmin2, rowmax2, colmin2,colmax2, ...
    min_color_val, max_color_val, min_color_val, max_color_val,...
    min_color_val, max_color_val, window_side_length, false);

%plot_raw_data(x2loc, y2loc)
%% Camera 3
%import data
load cam3_1.mat;
[height3_1 width3_1 rgb3_1 num_frames3_1] = size(vidFrames3_1);
% align timing with falling of the can
vidFrames3_1=vidFrames3_1(:,:,:,14:num_frames3_1);
```

```
[height3_1 width3_1 rgb3_1 num_frames3_1] = size(vidFrames3_1);

%For first search, search middle third of rows, middle third of cols
rowmin3 = floor(height3_1/3);
rowmax3 = height3_1 - floor(height3_1/3);
colmin3 = floor(width3_1/3);
colmax3 = width3_1 - floor(width3_1/3);
min_color_val = 235;
max_color_val = 255;
window_side_length = 20;
[x3loc, y3loc] = track_light(vidFrames3_1, height3_1, width3_1,...
    num_frames3_1, rowmin3, rowmax3, colmin3, colmax3,...
    min_color_val, max_color_val, min_color_val, max_color_val,...
    min_color_val, max_color_val, window_side_length, false);

%plot_raw_data(x3loc, y3loc)
%% make 6xtime Matrix
%matrix_width = min([num_frames1_1; num_frames2_1; num_frames3_1]);
matrix_width = 200;
case1_matrix = [x1loc(1:matrix_width);y1loc(1:matrix_width); ...
    x2loc(1:matrix_width); y2loc(1:matrix_width); x3loc(1:matrix_width)...
    ;y3loc(1:matrix_width)];
%% SVD
X = case1_matrix;
normalized = find_svd(X);

%% CASE 2
clear all; close all;

%% Camera 1
clear all; close all;
load cam1_2.mat;
[height1_2 width1_2 rgb1_2 num_frames1_2] = size(vidFrames1_2);
%align first trough
vidFrames1_2=vidFrames1_2(:,:,:,18:num_frames1_2);
[height1_2 width1_2 rgb1_2 num_frames1_2] = size(vidFrames1_2);

%search from half way through cols to 2/3 of wat through cols
colmin = floor(width1_2/2);
colmax = (width1_2 - floor(width1_2/3));
minc = 250;
maxc = 255;
rmin = minc;
rmax = maxc;
gmin = minc;
gmax = maxc;
bmin = minc;
bmax = maxc;


window_side_length = 40;
%look at bottom half of rows
rowmin = height1_2/2;
rowmax = height1_2;
```

```
[x1loc, y1loc] = track_light(vidFrames1_2, height1_2, width1_2,...
    num_frames1_2, rowmin, rowmax, colmin, colmax, ...
    rmin, rmax, gmin, gmax,...
    bmin, bmax, window_side_length, false);

%plot_raw_data(x1loc, y1loc)

%% Camera 2
load cam2_2.mat;
[height2_2 width2_2 rgb2_2 num_frames2_2] = size(vidFrames2_2);

rowmin2 = 1;
rowmax2 = height2_2;
%search only middle third of columns
colmin2 = floor(width2_2/3);
colmax2 = width2_2 - floor(width2_2/3);
%look for white pixel
minc = 240;
maxc = 255;
rmin = minc;
rmax = maxc;
gmin = minc;
gmax = maxc;
bmin = minc;
bmax = maxc;
window_side_length = 70;
[x2loc, y2loc] = track_light(vidFrames2_2, height2_2, width2_2,...
    num_frames2_2, rowmin2, rowmax2, colmin2,colmax2, ...
    rmin, rmax, gmin, gmax,...
    bmin, bmax, window_side_length, false);

%plot_raw_data(x2loc, y2loc)
%% Camera 3
load cam3_2.mat;
[height3_2 width3_2 rgb3_2 num_frames3_2] = size(vidFrames3_2);

%align first trough
vidFrames3_2=vidFrames3_2(:,:,:,20:num_frames3_2);
[height3_2 width3_2 rgb3_2 num_frames3_2] = size(vidFrames3_2);

%For first search, search middle third of rows, middle third of cols
rowmin3 = floor(height3_2/3);
rowmax3 = height3_2 - floor(height3_2/3);
colmin3 = floor(width3_2/3);
colmax3 = width3_2 - floor(width3_2/3);
%look for white pixeks
minc = 240;
maxc = 255;
rmin = minc;
rmax = maxc;
gmin = minc;
gmax = maxc;
bmin = minc;
bmax = maxc;
```

```
window_side_length = 70;
[x3loc, y3loc] = track_light(vidFrames3_2, height3_2, width3_2,...
    num_frames3_2, rowmin3, rowmax3, colmin3,colmax3, ...
    rmin, rmax, gmin, gmax,...
    bmin, bmax, window_side_length, false);

%plot_raw_data(x3loc, y3loc)
%% make 6xtime Matrix
%matrix_width = min([num_frames1_1; num_frames2_1; num_frames3_1]);
matrix_width = 200;
case2_matrix = [x1loc(1:matrix_width);y1loc(1:matrix_width); ...
    x2loc(1:matrix_width); y2loc(1:matrix_width); x3loc(1:matrix_width)...
    ;y3loc(1:matrix_width)];
%% SVD
X = case2_matrix;
normalized = find_svd(X);

%% CASE 3
clear all; close all;

%% Camera 1
clear all; close all;
load cam1_3.mat;
[height1_3 width1_3 rgb1_3 num_frames1_3] = size(vidFrames1_3);
%align first trough
vidFrames1_3=vidFrames1_3(:,:,:,9:num_frames1_3);
[height1_3 width1_3 rgb1_3 num_frames1_3] = size(vidFrames1_3);

%search from 1/3 way through cols to 1/2 of way through cols
colmin = floor(width1_3/3);
colmax = (width1_3 - floor(width1_3/2));

%look for pink pixel
rmin = 240;
rmax = 255;
gmin = 19;
gmax = 200;
bmin = 130;
bmax = 205;


window_side_length = 40;
%look at bottom half of rows
rowmin = height1_3/2;
rowmax = height1_3;
[x1loc, y1loc] = track_light(vidFrames1_3, height1_3, width1_3,...
    num_frames1_3, rowmin, rowmax, colmin, colmax, ...
    rmin, rmax, gmin, gmax,...
    bmin, bmax, window_side_length, false);

%plot_raw_data(x1loc, y1loc)

%% Camera 2
load cam2_3.mat;
```

```
[height2_3 width2_3 rgb2_3 num_frames2_3] = size(vidFrames2_3);

%search only bottom half of rows
rowmin2 = height2_3/2;
rowmax2 = height2_3;
%search only middle third of columns
colmin2 = floor(width2_3/3);
colmax2 = width2_3 - floor(width2_3/3);
%look for pink pixel
rmin = 240;
rmax = 255;
gmin = 19;
gmax = 200;
bmin = 130;
bmax = 205;
window_side_length = 40;
[x2loc, y2loc] = track_light(vidFrames2_3, height2_3, width2_3,...
    num_frames2_3, rowmin2, rowmax2, colmin2,colmax2, ...
    rmin, rmax, gmin, gmax,...
    bmin, bmax, window_side_length, false);

%plot_raw_data(x2loc, y2loc)
%% Camera 3
load cam3_3.mat;
[height3_3 width3_3 rgb3_3 num_frames3_3] = size(vidFrames3_3);

%align first trough
vidFrames3_3=vidFrames3_3(:,:,:,4:num_frames3_3);
[height3_3 width3_3 rgb3_3 num_frames3_3] = size(vidFrames3_3);

%For first search, search middle third of rows, middle third of cols
rowmin3 = floor(height3_3/3);
rowmax3 = height3_3 - floor(height3_3/3);
colmin3 = floor(width3_3/3);
colmax3 = width3_3 - floor(width3_3/3);
%look for pink pixel
rmin = 240;
rmax = 255;
gmin = 19;
gmax = 200;
bmin = 130;
window_side_length = 70;
[x3loc, y3loc] = track_light(vidFrames3_3, height3_3, width3_3,...
    num_frames3_3, rowmin3, rowmax3, colmin3,colmax3, ...
    rmin, rmax, gmin, gmax,...
    bmin, bmax, window_side_length, false);

%plot_raw_data(x3loc, y3loc)
%% make 6xtime Matrix
%matrix_width = min([num_frames1_1; num_frames2_1; num_frames3_1]);
matrix_width = 200;
case3_matrix = [x1loc(1:matrix_width);y1loc(1:matrix_width); ...
    x2loc(1:matrix_width); y2loc(1:matrix_width); x3loc(1:matrix_width)...
    ;y3loc(1:matrix_width)];
```

```
%% SVD
X = case3_matrix;
normalized = find_svd(X);

%% Case 4
close all;
clear all;
%% Camera 1
clear all; close all;
load cam1_4.mat;
[height1_4 width1_4 rgb1_4 num_frames1_4] = size(vidFrames1_4);

%search from 1/2 way through cols to 2/3 of way through cols
colmin = floor(width1_4/2);
colmax = (width1_4 - floor(width1_4/3));

%look for pink pixel
rmin = 240;
rmax = 255;
gmin = 19;
gmax = 200;
bmin = 130;
bmax = 205;


window_side_length = 40;
%look at bottom half of rows
rowmin = height1_4/2;
rowmax = height1_4;
[x1loc, y1loc] = track_light(vidFrames1_4, height1_4, width1_4,...
    num_frames1_4, rowmin, rowmax, colmin, colmax, ...
    rmin, rmax, gmin, gmax,...
    bmin, bmax, window_side_length, false);

%plot_raw_data(x1loc, y1loc)

%% Camera 2
load cam2_4.mat;
[height2_4 width2_4 rgb2_4 num_frames2_4] = size(vidFrames2_4);
%align first peak
vidFrames2_4=vidFrames2_4(:,:,:,10:num_frames2_4);
[height2_4 width2_4 rgb2_4 num_frames2_4] = size(vidFrames2_4);

%search only bottom half of rows
rowmin2 = height2_4/2;
rowmax2 = height2_4;
%search only middle third of columns
colmin2 = floor(width2_4/3);
colmax2 = width2_4 - floor(width2_4/3);
%look for pink pixel
rmin = 240;
rmax = 255;
gmin = 19;
gmax = 200;
```

```
bmin = 130;
bmax = 205;
window_side_length = 40;
[x2loc, y2loc] = track_light(vidFrames2_4, height2_4, width2_4,...
    num_frames2_4, rowmin2, rowmax2, colmin2,colmax2, ...
    rmin, rmax, gmin, gmax,...
    bmin, bmax, window_side_length, false);

%plot_raw_data(x2loc, y2loc)

%% Camera 3
load cam3_4.mat;
[height3_4 width3_4 rgb3_4 num_frames3_4] = size(vidFrames3_4);

%align first peak
vidFrames3_4=vidFrames3_4(:,:,:,4:num_frames3_4);
[height3_4 width3_4 rgb3_4 num_frames3_4] = size(vidFrames3_4);

%For first search, search middle third of rows, bottom half of cols
rowmin3 = floor(height3_4/3);
rowmax3 = height3_4 - floor(height3_4/3);
colmin3 = floor(width3_4/2);
colmax3 = width3_4;
% %look for white pixel
minc = 200;
maxc = 255;
rmin = minc;
rmax = maxc;
gmin = minc;
gmax = maxc;
bmin = minc;
bmax = maxc;
% rmin = 240;
% rmax = 255;
% gmin = 100;
% gmax = 200;
% bmin = 130;
% bmax = 205;
window_side_length = 70;
[x3loc, y3loc] = track_light(vidFrames3_4, height3_4, width3_4,...
    num_frames3_4, rowmin3, rowmax3, colmin3,colmax3, ...
    rmin, rmax, gmin, gmax,...
    bmin, bmax, window_side_length, false);

%plot_raw_data(x3loc, y3loc)

%% make 6xtime Matrix
%matrix_width = min([num_frames1_1; num_frames2_1; num_frames3_1]);
matrix_width = 200;
case4_matrix = [x1loc(1:matrix_width);y1loc(1:matrix_width); ...
    x2loc(1:matrix_width); y2loc(1:matrix_width); x3loc(1:matrix_width)...
    ;y3loc(1:matrix_width)];
%% SVD
X = case4_matrix;
```

```
normalized = find_svd(X);


%% Functions
function [pixel_color_in_range] = in_range(pixr, pixg, pixb, rmin, rmax,...
    gmin, gmax, bmin, bmax)
pixel_color_in_range = false;
if (pixr >= rmin) && (pixg >= gmin)&& (pixb >= bmin) && (pixr <= rmax)
    pixel_color_in_range = true;
end
end

function [] = plot_raw_data(xvec, yvec)
figure()
allvals = [xvec; yvec];
min_all = min(allvals);
min_y = min_all(1) - 10;
max_all = max(allvals);
max_y = max_all(1) + 10;
plot(1:length(xvec), xvec(:,1), "Linewidth", 1);
hold on;
plot(1:length(yvec), yvec(:,1), "Linewidth", 1);
set(gca, 'Ylim',[min_y, max_y], 'Fontsize', 16);
xlabel('Frame')
ylabel('Coordinate')
%legend()

legend('X Coordinate','Y Coordinate')
end

function [normalized_energy] = find_svd(X)
%% SVD
[m,n] = size(X); %data size
mn = mean(X,2); %row means
X = X - repmat(mn,1,n); %subtract mean

%u is principal directions, sigma is most important values, v is time
%series in each direction
[uhat,shat,v] = svd(X'/sqrt(n-1), 'econ'); %perform reduced svd


% Pull out eigen values
eigVals = diag(shat);


% Plot the eigen functions
figure;
subplot(1, 6, 1); plot(uhat(:,1));title('First Unit Basis Vector');
xlabel('Frame');
subplot(1, 6, 2); plot(uhat(:,2)); title('Second Unit Basis Vector');
xlabel('Frame');
subplot(1, 6, 3); plot(uhat(:,3)); title('Third Unit Basis Vector');
xlabel('Frame');
subplot(1, 6, 4); plot(uhat(:,4));title('Fourth Unit Basis Vector');
```

```
xlabel('Frame');
subplot(1, 6, 5); plot(uhat(:,5)); title('Fifth Unit Basis Vector');
xlabel('Frame');
subplot(1, 6, 6); plot(uhat(:,6)); title('Sixth Unit Basis Vector');
xlabel('Frame');


% Plot the amount of each mode in each dimension
figure;
subplot(2, 3, 1); bar(v(:,1));
title('Importance of Unit Vector 1 in each Input');
xticks(1:1:6)
ylabel('Dot Product Result');
xticklabels({'Cam1X','Cam1Y','Cam2X','Cam2Y','Cam3X','Cam3Y'})
subplot(2, 3, 2); bar(v(:,2));
title('Importance of Unit Vector 2 in each Input');
xticks(1:1:6)
xticklabels({'Cam1X','Cam1Y','Cam2X','Cam2Y','Cam3X','Cam3Y'})
ylabel('Dot Product Result');
subplot(2, 3, 3); bar(v(:,3));
title('Importance of Unit Vector 3 in each Input');
xticks(1:1:6)
xticklabels({'Cam1X','Cam1Y','Cam2X','Cam2Y','Cam3X','Cam3Y'})
ylabel('Dot Product Result');
subplot(2, 3, 4); bar(v(:,4));
title('Importance of Unit Vector 4 in each Input');
xticks(1:1:6)
xticklabels({'Cam1X','Cam1Y','Cam2X','Cam2Y','Cam3X','Cam3Y'})
ylabel('Dot Product Result');
subplot(2, 3, 5); bar(v(:,5));
title('Importance of Unit Vector 5 in each Input');
xticks(1:1:6)
xticklabels({'Cam1X','Cam1Y','Cam2X','Cam2Y','Cam3X','Cam3Y'})
ylabel('Dot Product Result');
subplot(2, 3, 6); bar(v(:,6));
title('Importance of Unit Vector 6 in each Input');
xticks(0:1:6)
xticklabels({'', 'Cam1X','Cam1Y','Cam2X','Cam2Y','Cam3X','Cam3Y', ''})
ylabel('Dot Product Result');


normalized_energy = eigVals/sum(eigVals);

% The cumulative energy content for the m'th eigenvector is the
%sum of the energy content across eigenvalues 1:m
running_sum = 0;
num_modes_needed = 0;
i = 1;
while running_sum <= .80;
    running_sum = running_sum + normalized_energy(i);
    num_modes_needed = i;
    i = i+1;
end
fprintf('Number Modes Needed =%d \n', num_modes_needed)
```

```
485
486
487 bestrank1 = uhat(:,1)*eigVals(1)*v(:,1)';
488
489 figure;
490 subplot(3,1,1); plot(X'/sqrt(n-1)); title('Pre-Processed Input Data');
491 legend('cam1 X', 'cam1 Y', 'cam2 X', 'cam2 Y', 'cam3 X', 'cam3 Y')
492 bestrank = bestrank1;
493 i = 1;
494 while i < num_modes_needed
495     i = i + 1;
496     bestrank = bestrank + (uhat(:,2)*eigVals(i)*v(:,1)');
497 end
498 subplot(3, 1, 2); plot(bestrank1); title('Best Rank 1 Approximation of Data');
499 legend('cam1 X', 'cam1 Y', 'cam2 X', 'cam2 Y', 'cam3 X', 'cam3 Y')
500 subplot(3, 1, 3); plot(bestrank); title(sprintf('Best Rank %d Approximation of
        data', num_modes_needed));
501 xlabel('Frame');
502 legend('cam1 X', 'cam1 Y', 'cam2 X', 'cam2 Y', 'cam3 X', 'cam3 Y')
503
504 figure()
505 %eigenfunctions scaled to sigma
506 min_y = min(uhat(:,1)*eigVals(1));
507 max_y = max(uhat(:,1)*eigVals(1));
508 subplot(6, 1, 1); plot(uhat(:,1)*eigVals(1));title('First Unit Vector * First
    Sigma');
509 set(gca, 'Ylim',[min_y, max_y], 'Fontsize', 10);
510 subplot(6, 1, 2); plot(uhat(:,2)*eigVals(2)); title('Second Unit Vector *
    Second Sigma');
511 set(gca, 'Ylim',[min_y, max_y], 'Fontsize', 10);
512 subplot(6, 1, 3); plot(uhat(:,3)*eigVals(3)); title('Third Unit Vector * Third
        Sigma');
513 set(gca, 'Ylim',[min_y, max_y], 'Fontsize', 10);
514 subplot(6, 1, 4); plot(uhat(:,4)*eigVals(4));title('Fourth Unit Vector *
    Fourth Sigma');
515 set(gca, 'Ylim',[min_y, max_y], 'Fontsize', 10);
516 subplot(6, 1, 5); plot(uhat(:,5)*eigVals(5)); title('Fifth Unit Vector * Fifth
        Sigma');
517 set(gca, 'Ylim',[min_y, max_y], 'Fontsize', 10);
518 subplot(6, 1, 6); plot(uhat(:,6)*eigVals(6)); title('Sixth Unit Vector * Sixth
        Sigma');
519 set(gca, 'Ylim',[min_y, max_y], 'Fontsize', 10);
520 xlabel('Frame');
521 end
522
523 function [xloc, yloc] = track_light(vidframes, height, width,...
524     num_frames, rowmin, rowmax, colmin, colmax, rmin, rmax, gmin, gmax, ...
525     bmin, bmax, window_side_length, parameter_optimize_mode)
526
527 %camera location vectors
528 xloc = zeros(num_frames);
529 yloc = zeros(num_frames);
530
531 %iterate through each frame
```

```
for frame = 1 : num_frames
    thisframe = vidframes(:,:,:,frame);
    %only find one location per frame
    location_found = false;

    %look at rows within selected range (window centered on prev location)
    for row = rowmin:rowmax
        %look at cols within selected range
        %(window centered on prev location)
        for col = colmin:colmax
            %if the location hasn't already been found and the point is in
            %the window and close to white
            rpix = thisframe(row,col,1);
            gpix = thisframe(row,col,2);
            bpix = thisframe(row,col,3);
            if ((location_found) == false &&...
                    in_range(rpix, gpix, bpix, rmin, rmax, gmin, gmax,...
                    bmin,bmax) == true)
                %set this boolean to true (so take first white point found
                %in window to be the location)
                location_found = true;
                %save x and y locations in vectors
                xloc(frame) = col;
                yloc(frame) = row;
                %set window dimensions for next frame
                rowmin = row - window_side_length;
                rowmax = row + window_side_length;
                colmin = col - window_side_length;
                colmax = col + window_side_length;
                if rowmin < 1
                    rowmin = 1;
                end
                if colmin < 1
                    colmin = 1;
                end
                if rowmax > height
                    rowmax = height;
                end
                if colmax > width
                    colmax = width;
                end
            end
        end
    end

    %If optimize_parameters is true, show me the video of the overlayed
    %window in its proper location in each frame and tell me if any frames
    %didn't find a point in the window:
    if parameter_optimize_mode == true
        for row = 1:height
            for col = 1:width
                if (row > rowmin && (row < rowmax)...
                        && (col > colmin)) && (col < colmax)
                    thisframe(row,col,1) = 255;
```

```
586                     thisframe(row,col,2) = 0;
587                     thisframe(row,col,3) = 0;
588                 end
589             end
590         end
591         if location_found == false
592             fprintf("No location found in frame %d\n", frame)
593         end
594         imshow(thisframe); drawnow
595     end
596
597     %if no location found, set this location to be the same as the location
598     %in the last frame
599     if location_found == false
600         xloc(frame) = xloc(frame-1);
601         yloc(frame) = yloc(frame-1);
602     end
603 end
604
605 %function end
606 end
```

Appendix C Full Plots and Tables

Case	Basis Vector	Principal Component	Percent of Overall Energy in Matrix	Cumulative Sum of Energy in Matrix
1	1	85.98	74.72	74.72
1	2	12.82	11.14	85.86
2	1	72.72	41.12	41.12
2	2	37.95	21.50	62.62
2	3	23.95	13.57	76.19
2	4	19.10	10.82	87.01
3	1	51.08	39.48	39.46
3	2	31.00	23.96	63.44
3	3	15.92	12.31	75.75
3	4	15.17	11.72	87.47
4	1	54.05	38.92	38.92
4	2	37.89	27.28	66.2
4	3	15.19	10.94	77.14
4	4	13.34	9.60	86.74

Table 1: Percent energy captured by basis vectors required to represent 80% of the energy in the Matrix. Condition 1 required a 2-rank approximation to capture 80% of the energy in the Matrix. Conditions 2, 3 and 4 all required rank 4 approximations.

Case 1: Ideal

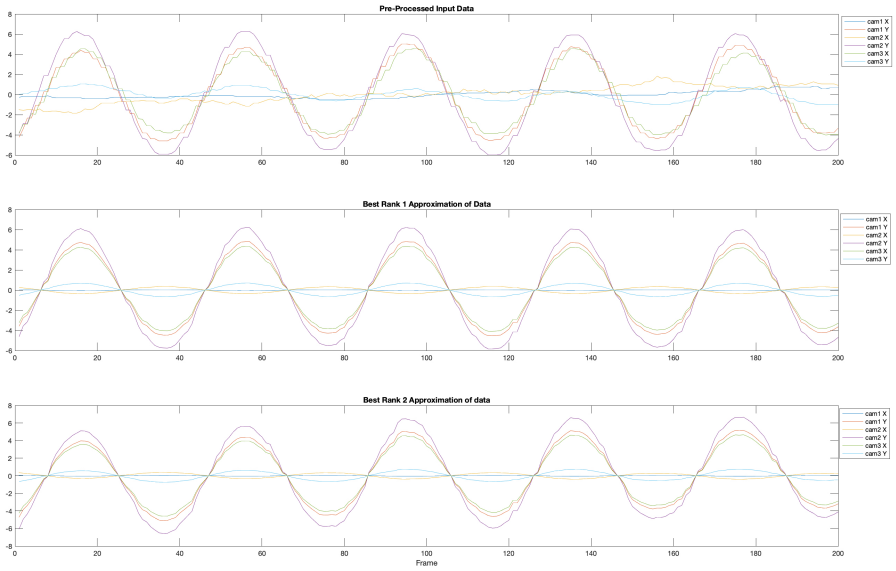

Figure 3: Case 1. Top: Data in Matrix X after mean subtraction and setting variance to unity. Middle and Bottom are one-rank and two-rank approximations of X respectively.

Case 1: Ideal

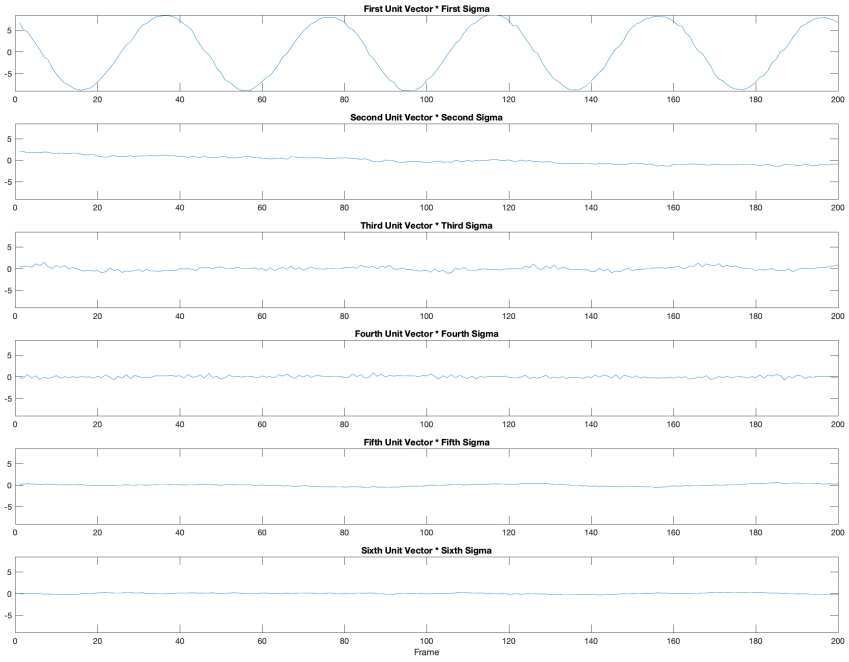

Figure 4: Case 1. Standard basis vectors in order from most (top) to least (bottom) important, scaled to their respective principal component values

Case 2: Noisy

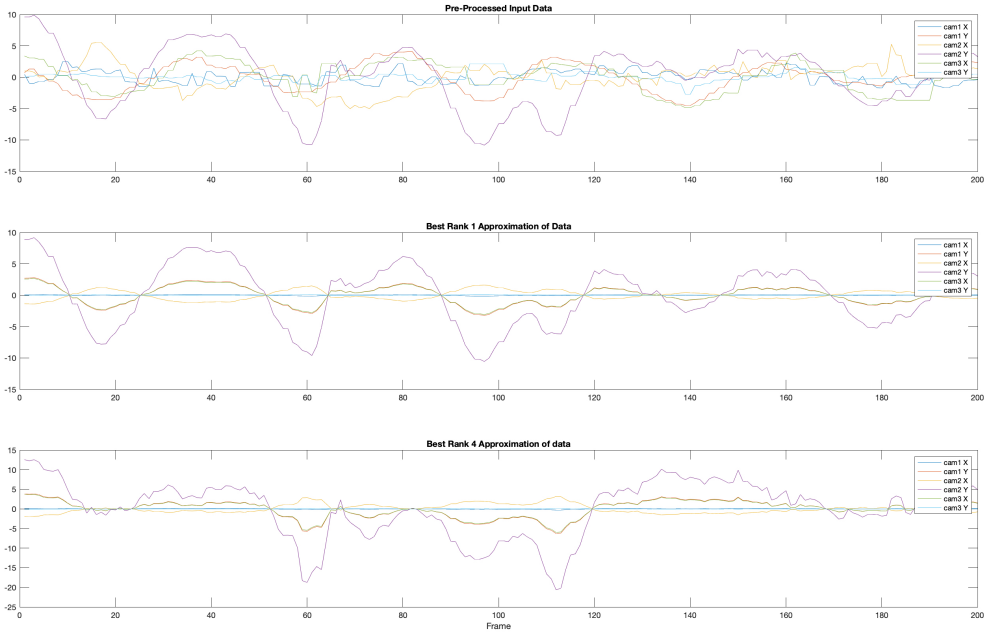

Figure 5: Case 2. Top: Data in Matrix X after mean subtraction and setting variance to unity. Middle and Bottom are one-rank and four-rank approximations of X respectively.

Case 2: Noisy

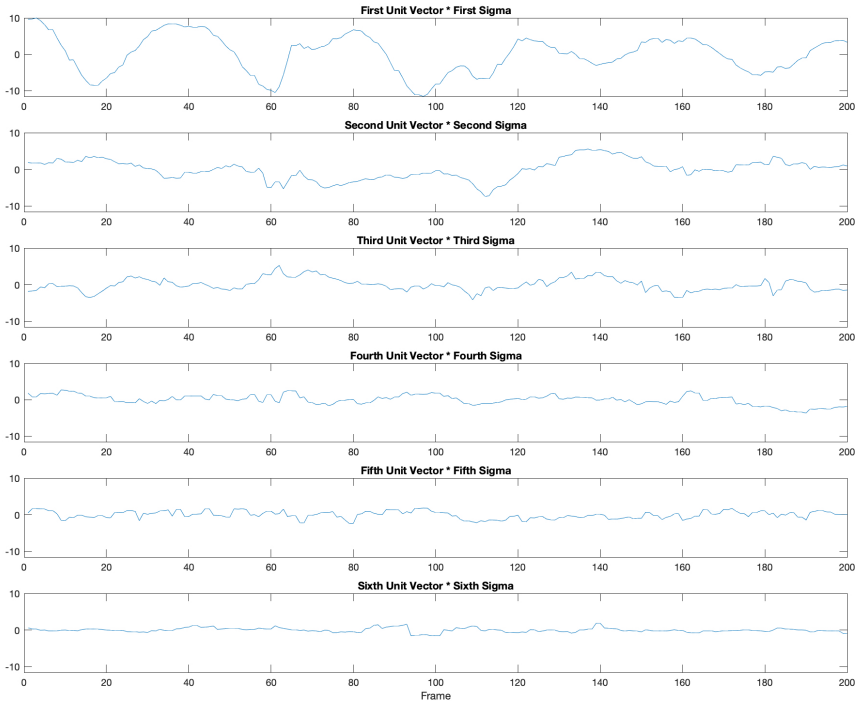

Figure 6: Case 2. Standard basis vectors in order from most (top) to least (bottom) important, scaled to their respective principal component values

Case 3: Horizontal Displacement

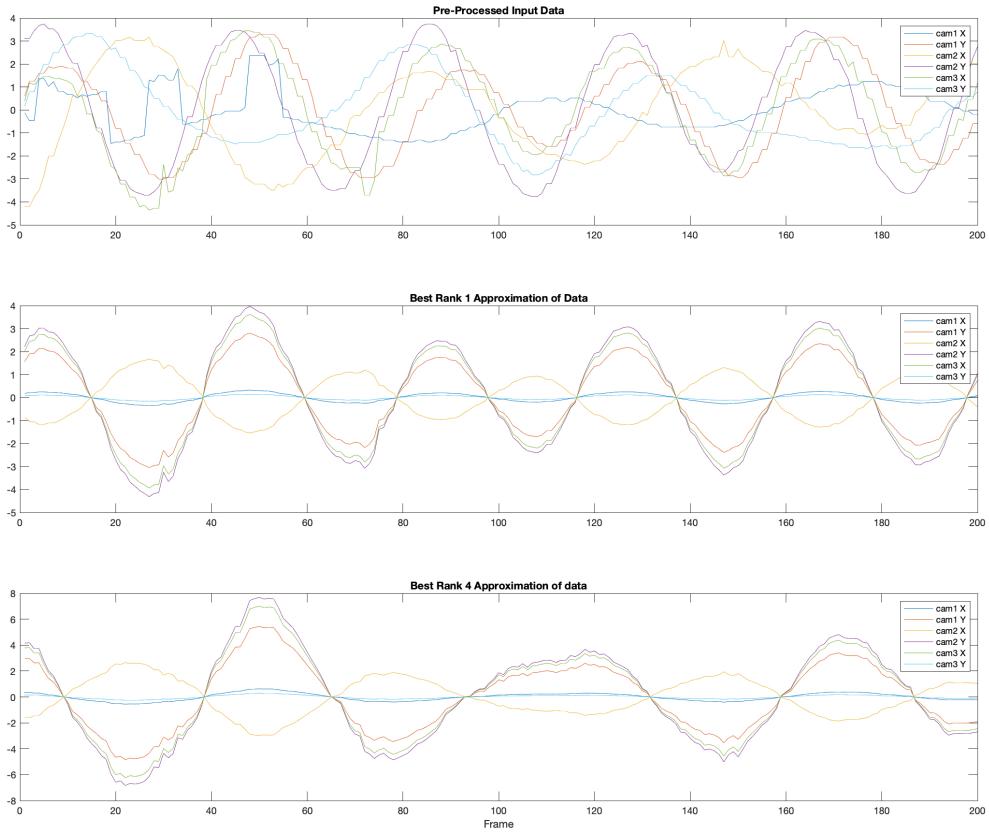

Figure 7: Case 3. Top: Data in Matrix X after mean subtraction and setting variance to unity. Middle and Bottom are one-rank and four-rank approximations of X respectively.

Case 3: Horizontal Displacement

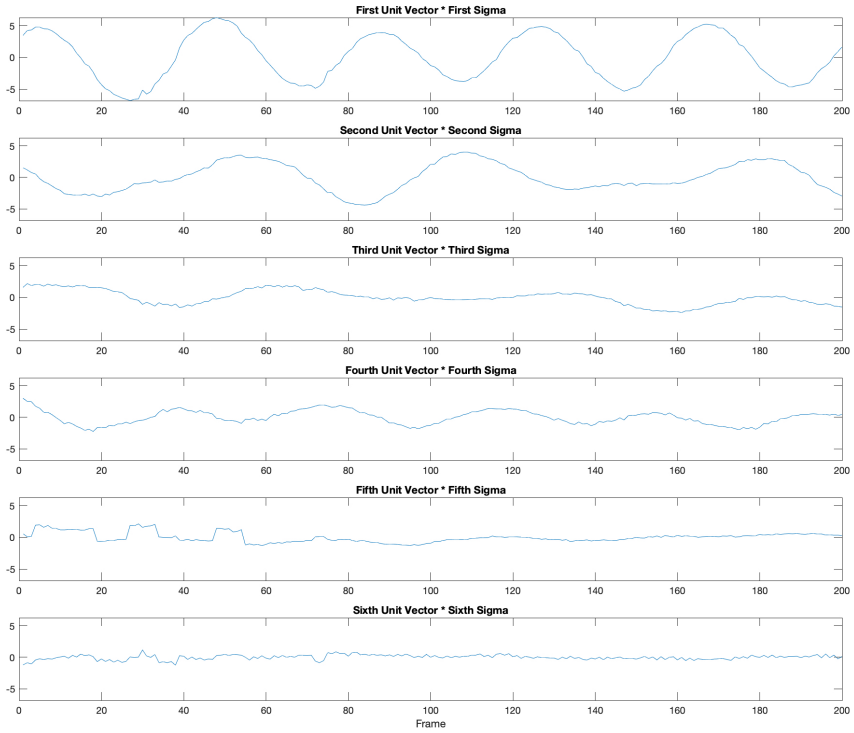

Figure 8: Case 3. Standard basis vectors in order from most (top) to least (bottom) important, scaled to their respective principal component values

Case 4: Horizontal Displacement and Rotation

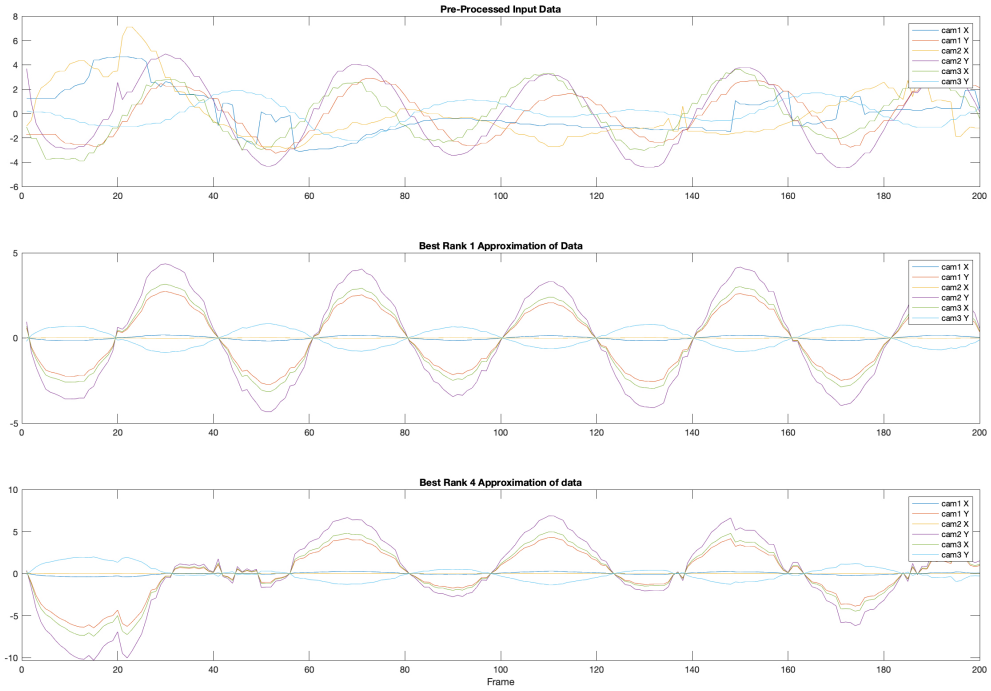

Figure 9: Case 4. Top: Data in Matrix X after mean subtraction and setting variance to unity. Middle and Bottom are one-rank and four-rank approximations of X respectively.

Case 4: Horizontal Displacement and Rotation

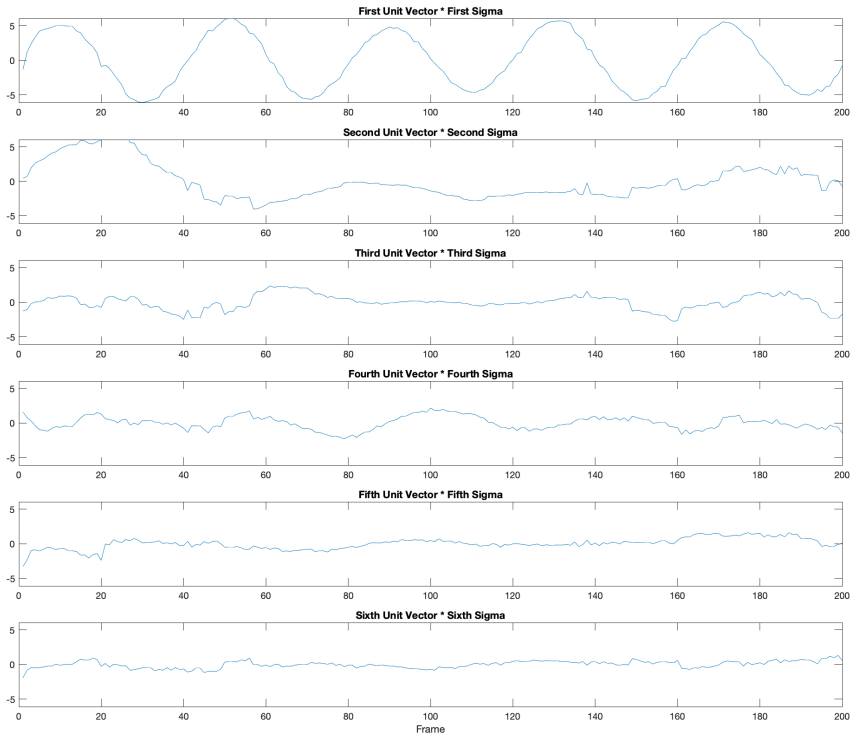

Figure 10: Case 4. Standard basis vectors in order from most (top) to least (bottom) important, scaled to their respective principal component values

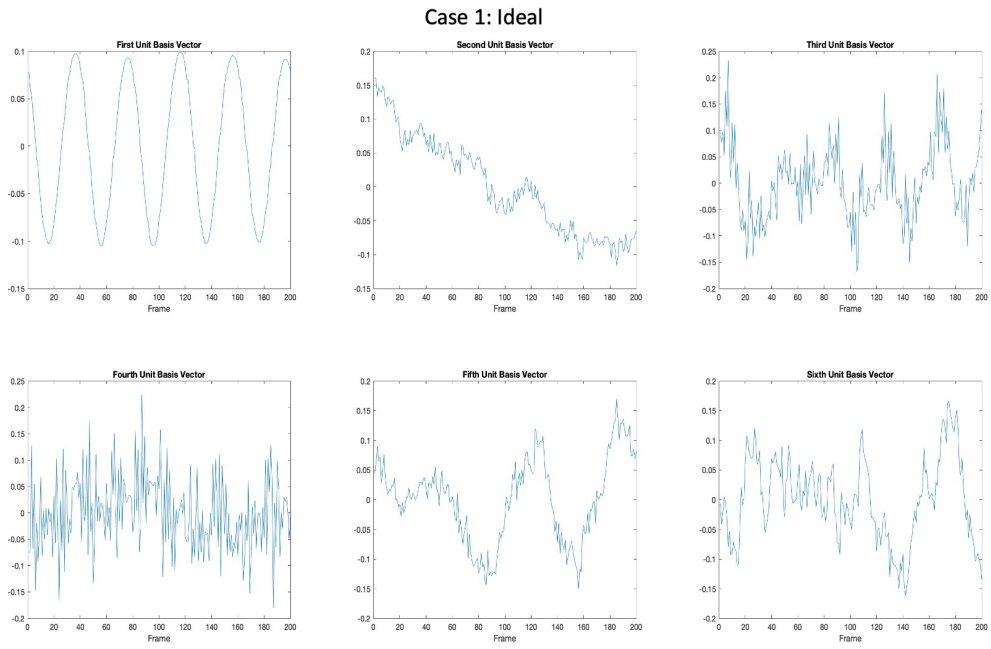

Figure 11: Unscaled Unit Basis Vectors in order from most important (upper left) to least important (lower right).

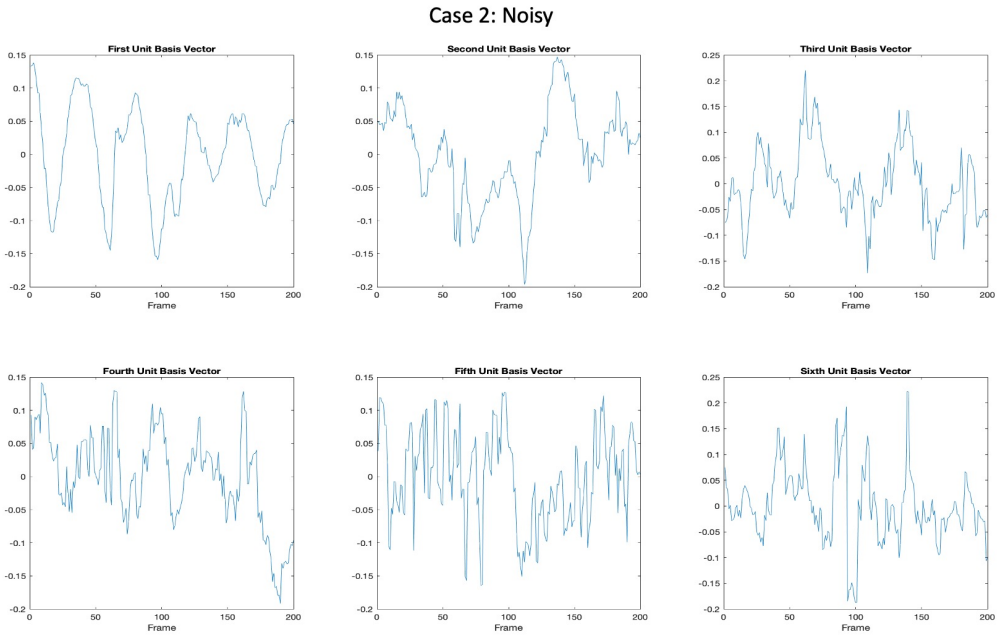

Figure 12: Unscaled Unit Basis Vectors in order from most important (upper left) to least important (lower right).

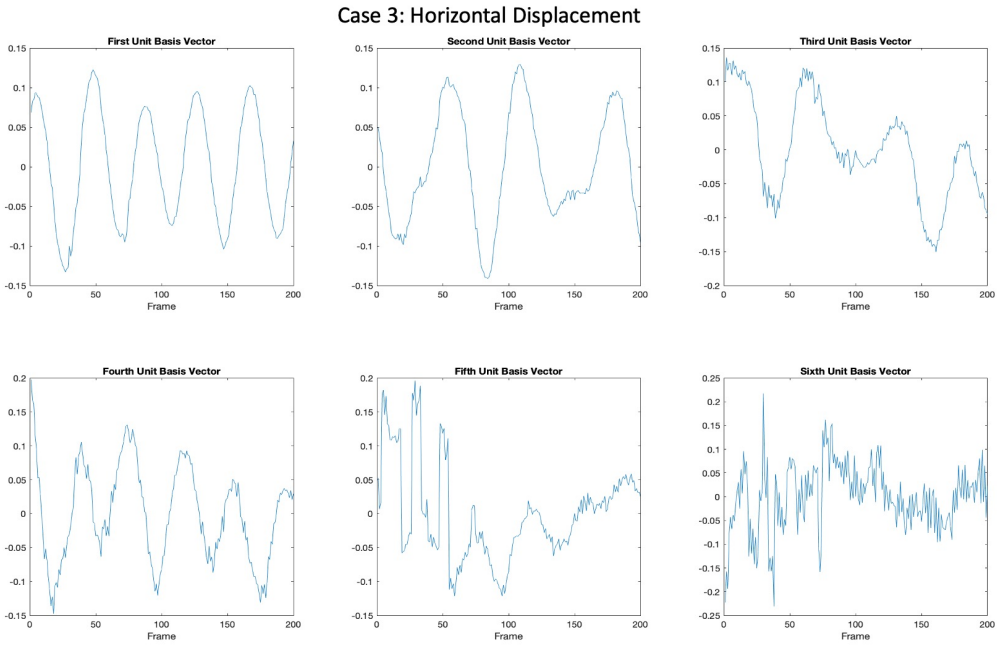

Figure 13: Unscaled Unit Basis Vectors in order from most important (upper left) to least important (lower right).

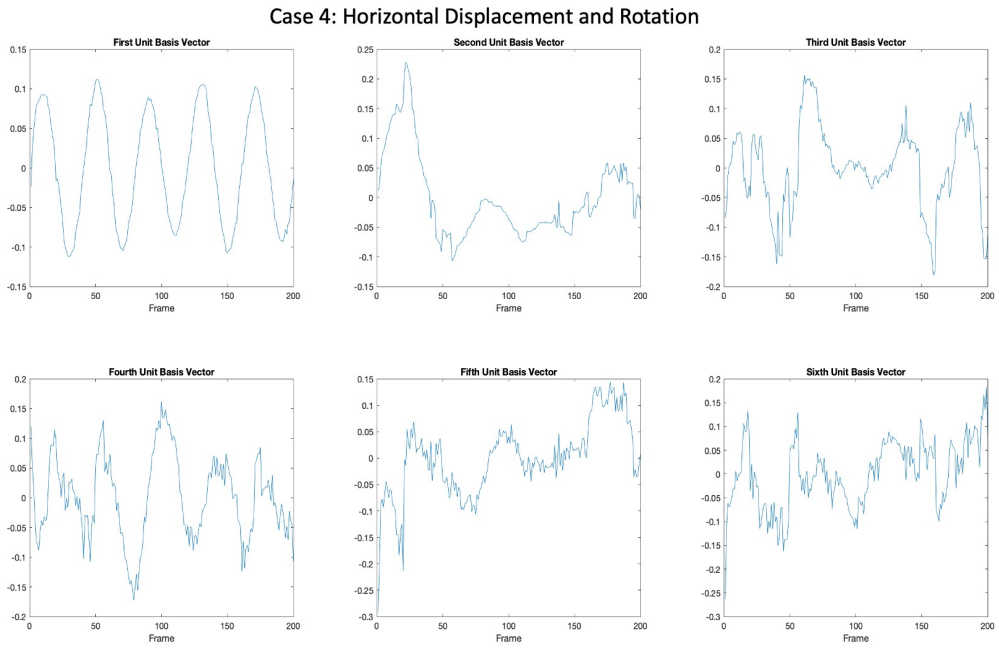

Figure 14: Unscaled Unit Basis Vectors in order from most important (upper left) to least important (lower right).