# AMATH 482 Homework 3

Haley Riggs

February 18, 2021

**Abstract**

The singular value decomposition (SVD) and principal component analysis (PCA) are incredibly useful tools for many applications of math. Through their use, it is easy to find trends within data and calculate low-dimensional approximations of data. In this report, we use the SVD and PCA to analyze the motion of a can on a string for four different test cases.

## 1 Introduction and Overview

In this assignment, we were given movie files of a paint can on a string. There were four different test cases, an ideal case, a noisy case, a case with horizontal displacement, and a case with horizontal displacement and rotation. Three cameras recorded videos associated with each case. One of these videos was rotated 90 degrees. However, no further data manipulation was necessary, as the orientation is irrelevant to the PCA.

After cropping the movie to remove most of the background and focus in on the can, the position data for the can was extracted. Then, principal component analysis was applied to pull out the principal components of the data. This assignment discusses just one example of the vast applications of principal component analysis (PCA).

## 2 Theoretical Background

Principal component analysis (PCA) is founded upon the singular value decomposition (SVD). Matrices, geometrically, are essentially linear transformations. When a vector is multiplied by a matrix, it is transformed into a new vector through a rotation and/or scaling. Consider $2 \times 2$ matrices for the following discussion.

### 2.1 The Singular Value Decomposition (SVD)

Orthogonal matrices have columns and rows that are all orthogonal. The inverse of an orthogonal matrix is equal to its transpose. Unitary matrices are matrices with complex entries whose inverses are equal to their conjugate transpose. Orthogonal matrices can also be thought of as unitary. Neither orthogonal nor unitary matrices stretch or compress vectors. Another way to put this fact is that for any unitary or orthogonal matrix $\mathbf{A}$, the Euclidean (or 2-norm) of $\mathbf{Ax}$ for a real-valued vector $\mathbf{x}$ is equal to the the Euclidean norm of simply the vector. This quantity is given by

$$||\mathbf{x}||_2 = |\mathbf{Ax}||_2 = \sqrt{\sum_{n=1}^{N} |x_n|^2}. \tag{1}$$

Matrices that scale vectors are diagonal, with the entries on the diagonal not equalling zero. Often, only positive entries for the diagonal are considered; negative entries correspond to stretching or compressing the vector and flipping it in the opposite direction. Diagonal entries greater than one correspond to stretching the vector and entries between zero and one, exclusive, correspond to compression.

It can be shown that multiplying a vector by a $2 \times 2$ matrix will change circles into ellipses, through stretching and compressing the circle in different directions. Therefore, if we consider multiplying the unit circle by a matrix, we will get an ellipse. The principal semiaxes of the ellipse can be labeled as $\sigma_1\mathbf{u}_1$ and

$\sigma_2\mathbf{u}_2$, where $\sigma_1$ and $\sigma_2$ represent how long the vectors are and $\mathbf{u}_1$ and $\mathbf{u}_2$ are the unit vectors pointing in the directions of these axes. If this is the case, we know that some unit vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ exist such that

$$\mathbf{A}\mathbf{v}_1 = \sigma_1\mathbf{u}_1, \qquad \mathbf{A}\mathbf{v}_2 = \sigma_2\mathbf{u}_2 \tag{2}$$

Putting these equations into a corresponding matrix equation gives us

$$\mathbf{AV} = \mathbf{U\Sigma} \tag{3}$$

where the columns of $\mathbf{V}$ are $\mathbf{v}_1$ and $\mathbf{v}_2$ ($\mathbf{V}$ is unitary), the columns of $\mathbf{U}$ are $\mathbf{u}_1$ and $\mathbf{u}_2$ ($\mathbf{U}$ is unitary), and $\mathbf{\Sigma}$ is a diagonal matrix with diagonal entries $\sigma_j$. Noting the properties of these matrices, Equation 3 can be rewritten to form a decomposition of $\mathbf{A}$:

$$\mathbf{A} = \mathbf{U\Sigma V}^* \tag{4}$$

Thankfully, this theory can be applied to matrices that are not $2\times2$ *or even square*. If $\mathbf{A}$ is an $m\times n$ matrix, multiplying an $n$-dimensional unit sphere by it will produce a hyperellipse that exists in $m$-dimensional space. The hyperellipse will be $n$-dimensional if $m \geq n$ and $\mathbf{A}$ is a full rank matrix. Instead of two principal subaxes as in the previous case, we now have $n$ principal semiaxes whose lengths we can denote as $\sigma_1, \sigma_2, \ldots, \sigma_n$ that correspond to unit vectors $\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_n}$ pointing in the direction of the semiaxes. Once again, there exist vectors $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}$ such that

$$\mathbf{A}\mathbf{v}_1 = \sigma_1\mathbf{u}_1, \quad \mathbf{A}\mathbf{v}_2 = \sigma_2\mathbf{u}_2, \ldots, \quad \mathbf{A}\mathbf{v}_n = \sigma_n\mathbf{u}_n \tag{5}$$

We can write this in matrix form and solve for $\mathbf{A}$, where each matrix has a similar meaning to what was discussed in the previous case (Equation 3). Our result, Equation 6, is what is known as the **reduced SVD**.

$$\mathbf{A} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\mathbf{V}^* \tag{6}$$

The full SVD is similar to this, except additional orthonormal columns are added to $\hat{\mathbf{U}}$ to make it an $m \times m$ square, resulting in a matrix $\mathbf{U}$, and additional rows of zeros are added to $\hat{\mathbf{\Sigma}}$ to make it $m \times n$. $\mathbf{V}$ is an $n \times n$ square matrix. Therefore, $\mathbf{A}$ can be represented as in Equation 4.

Essentially, the SVD is composed of a rotation, a stretch/compression, and another rotation. The **singular values** of $\mathbf{A}$ are the diagonal entries of $\mathbf{\Sigma}$, $\sigma_n$, the columns of $\mathbf{U}$, vectors $\mathbf{u_n}$ are the **left singular vectors** of $\mathbf{A}$, and the columns of $\mathbf{V}$, vectors $\mathbf{v_n}$ are the **right singular vectors** of $\mathbf{A}$.

It should be noted that there exists an SVD for every matrix $\mathbf{A}$. Additionally, the singular values are always real numbers that are greater than 0 and are uniquely determined. Inside of $\mathbf{\Sigma}$, the singular values along the diagonal are ordered from greatest to least and the rank of $\mathbf{A}$ is equal to the number of nonzero singular values. Also, if $r$ is the rank of $\mathbf{A}$, the range of $\mathbf{A}$ can be represented by $\{\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_r}\}$ and the basis for the null space of $\mathbf{A}$ is given by $\{\mathbf{v_r} + 1, \ldots, \mathbf{v_n}\}$.

The SVD can be calculated with MATLAB's function `svd()`. Usually, however, it is unnecessary to derive the full SVD for a matrix; we only really need the reduced SVD. So, we use the `'econ'` option inside of MATLAB's `svd()` function.

The matrix $\mathbf{A}$ can be represented by the following equation, assuming $\mathbf{A}$ has rank $r$:

$$\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u_j}\mathbf{v_j}^* \tag{7}$$

In other words, $\mathbf{A}$ can be written as the sum of rank-one matrices. $\mathbf{u_j}\mathbf{v_j}^*$ is known as an *outer product* and will be the same size as $\mathbf{A}$.

Using Equation 7, it is possible to show that $\mathbf{A}$ can be approximated by

$$\mathbf{A_N} = \sum_{j=1}^{N} \sigma_j \mathbf{u_j}\mathbf{v_j}^* \tag{8}$$

where $N$ is between zero and $r$, inclusive. It can be shown that Equation 8 gives the best approximation of $\mathbf{A}$ of all matrices of rank $N$ or less. This low-rank approximation can help with performing *dimensionality* *reduction* or *low-dimensional approximations* of data, maintaining the most important information with the fewest number of dimensions of the data.

In order to calculate how much **energy** from $\mathbf{A}$ is contained in each rank-$N$ approximation, we can divide the sum of the squares of the singular values contained in the approximation by the sum of the squares of all of the singular values in $\mathbf{A}$. This is given by the equation

$$\text{energy}_N = \frac{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_N^2}{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_r^2} \tag{9}$$

## 2.2 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a method that uses the SVD to find trends within a set of data. In order to understand PCA, we need to consider some preliminary statistics.

If we consider a vector $\mathbf{a} = [a_1 \;\; a_2 \;\; \ldots \;\; a_n]$, we know that the mean of the data in the vector can be given by

$$\mu = \frac{1}{n}\sum_{k=1}^{n} a_k \tag{10}$$

For PCA, we should subtract the mean, $\mu$, off of the data, thereby assuming that $\mu = 0$. We can also derive the variance, an unbiased estimator, which for our situation is equal to

$$\sigma^2 = \frac{1}{n-1}\mathbf{a}\mathbf{a}^{\mathbf{T}} \tag{11}$$

If instead of just one vector $\mathbf{a}$ of data, we have two, namely $\mathbf{a}$ and $\mathbf{b}$, which both have length $n$ and mean zero, we can calculate their **covariance** by

$$\sigma_{ab}^2 = \frac{1}{n-1}\mathbf{a}\mathbf{b}^{\mathbf{T}} \tag{12}$$

If this covariance is equal to zero, $\mathbf{a}$ and $\mathbf{b}$ are **uncorrelated** and therefore **statistically independent**. This means that possessing information about $\mathbf{a}$ does not reveal anything about $\mathbf{b}$; there is no redundancy in the data.

Now, considering multiple vectors of data, which can be put as rows inside a matrix $\mathbf{X}$, we can use a simple matrix multiplication to compute the variances and covariances between the rows of $\mathbf{X}$. This equation is given by

$$\mathbf{C_x} = \frac{1}{n-1}\mathbf{X}\mathbf{X}^{\mathbf{T}}, \tag{13}$$

where the diagonal entries of matrix $\mathbf{C_x}$ are the variances of each row of data inside $\mathbf{X}$ and the off-diagonal entries are the covariances. $\mathbf{C_x}$ is a square, symmetric matrix known as the **covariance matrix**.

In PCA, we aim to find a change of basis so that our variables are uncorrelated and therefore do not contain any redundant information. In order to discover which variables have the greatest variances, and therefore tell us the most information about the data, we can diagonalize $\mathbf{C_x}$, where all covariances are zero:

$$\mathbf{C_x} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^{\mathbf{-1}} \tag{14}$$

The principal components are the basis of eigenvectors, which are uncorrelated and orthogonal, inside of $\mathbf{V}$. The eigenvalues of $\mathbf{C_x}$ are the diagonal entries of $\boldsymbol{\Lambda}$ and are the corresponding variances.

It can be shown that

$$\mathbf{A} = \frac{1}{\sqrt{n-1}}\mathbf{X} \tag{15}$$

and therefore, if we let $\mathbf{U}$ be the orthogonal matrix composed of left-singular vectors and $\boldsymbol{\Sigma}$ contain the singular values on its diagonal with the rest of its entries being zero,

$$\mathbf{C_x} = \frac{1}{n-1}\mathbf{XX^T} = \mathbf{AA^T} = \mathbf{U\Sigma^2U^T} \tag{16}$$

From this analysis, we can note that the eigenvalues of the covariance matrix $\mathbf{C_x}$ are equal to the scaled singular values, squared.

Now in order to transform the data into the basis of the principal components, we must multiply the data matrix $\mathbf{X}$ by $\mathbf{U^{-1}} = \mathbf{U^T}$ on the left. We can show that the covariance of the resulting matrix is simply equal to $\mathbf{\Sigma^2}$, which has zeros as all of its off-diagonal entries and therefore shows that the variables contained in said resulting matrix are entirely uncorrelated.

# 3 Algorithm Implementation and Development

In this assignment, position data for a can moving on a string was extracted from video files and then ran through PCA to obtain the principal components of the scenario. Four different tests were performed. In each, three different cameras, each from different positions in relation to the phenomenon, recorded what was happening.

First, the data from the given test case was loaded into MATLAB and explored through the use of the function `implay()`. After watching the video, the corresponding video frame was cropped to remove irrelevant background images and focus in on the can and string. The cropped region was discovered manually through essentially a trial and error procedure. This was repeated for each video corresponding to the current test.

Next, the position of the can was extracted through tracking the most red component in the cropped video. (When the videos were recorded, a red laser was shining on the can). This was executed through the use of a for loop which ran from one to the size of the time component of the cropped video frames tensor. For each time-step (i.e. frame), the row and column data for the red color channel was extracted. The index of the element with the maximum value was obtained and then used to find the corresponding position coordinates inside the video frames tensor. This data was then stored as a row inside of a position data matrix. This process was repeated for all three cropped videos relevant to the current test.

At this point, the mean of each set of results was set to zero. The mean for the $x$ and $y$ coordinates for each camera was subtracted from the data. Then, three plots were created, each containing the horizontal data and the vertical data for each camera for the given test.

After this, the $x$-coordinate of the first "peak" of the sinusoidal data (which is explained further in the next paragraph) was obtained. In future tests, this became increasingly difficult to find. This peak was at a slightly different location for each video. The data vectors were then trimmed accordingly, to line up their initial peaks. At this point, some of the vectors were still longer than the others. The vectors were trimmed again to make them all the same size.

For the first test, at least, the range of the horizontal data was relatively constant while the range of the vertical was more obviously sinusoidal. This made sense, as the can was moving primarily in the vertical direction. Interestingly enough, the horizontal data corresponded to the second column in the position data and the vertical data corresponded to the first column. This is because the video frame data was given in the form of a four-dimensional tensor of row data versus column data versus color data versus time data.

The next step was to construct a matrix of the trimmed position data, letting the position data vectors be the rows of this matrix in the order $x$-coordinate for camera 1, $y$-coordinate for camera 1, $x$-coordinate for camera 2, and so on. The reduced SVD and the energies added by each $\sigma$ value were calculated. (The energy added by each consequent $\sigma$ can be calculated by dividing the square of the $\sigma$ of interest by the sum of the squares of all the $\sigma$ values.)

The entire processed discussed thus far was then repeated for each test case. Afterwards, the number of $\sigma$s needed for at least 95% energy was calculated for each test case. A for loop looping over all the $\sigma$ values, summing up the energy associated with that $\sigma$ value and those prior until the 95% energy threshold was met was created. The total energies and number of required $\sigma$ values to reach at least 95% energy for each test were stored in vectors. Then, plots of the columns of $\mathbf{V}$ and the $\sigma$ energies for each test case were created. All of the MATLAB code for the methods discussed in this section is included in Appendix B.

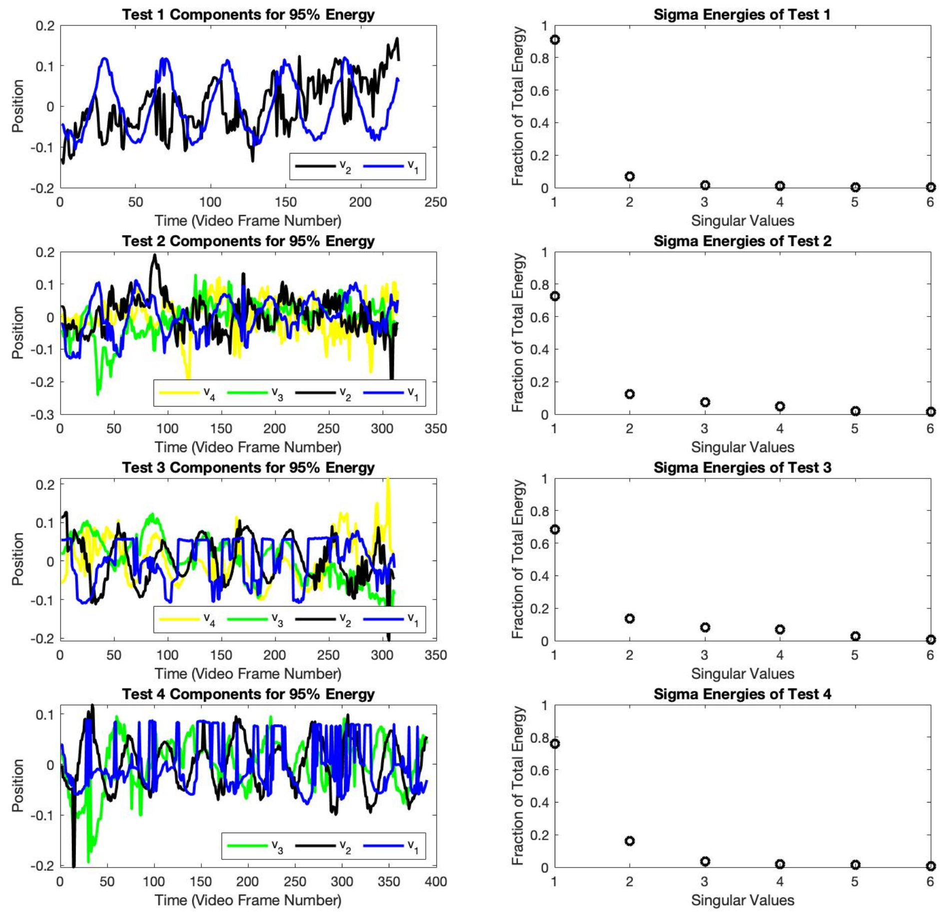


Figure 1: Here are the components and $\sigma$ energies for each test case to retain 95% energy.

# 4 Computational Results

As mentioned previously, three cameras recorded videos of a moving can on a string for each of four different test cases. The first case was an idealized up-and-down motion, the second test was comprised of noisy recordings of an up-and-down motion (the cameras were violently shaking), the third test case included horizontal displacement, and the fourth had both horizontal displacement and rotation of the can.

Images of the position data are included in Appendix C. It can be observed for the first case, at least, that the vertical movement of the can is oscillatory, which is intuitive as the can moves up and down. The horizontal movement is relatively constant with mild sinusoidal movement. This, too, is intuitive as the can's primary movement is in the up-and-down, or vertical, direction.

In Figure 1, there is a depiction of the components and $\sigma$ energies to obtain 95% energy. As discussed in Section 3, certain code was run to calculate the number of singular values needed for each test case in order to capture 95% of the phenomenon's energy. For Test 1, only two singular values were necessary to reach the 95% threshold. In accordance to this, the first two columns of the $\mathbf{V}$ matrix, labeled $v_1$ and $v_2$ on the graph, resulting from the SVD are plotted. (Note that all components are plotted in reverse order, i.e. the column corresponding to the last singular value necessary first and the column corresponding to the first $\sigma$ last, in order to more prominently show the most important data trends.)

From the plot in the upper left corner of Figure 1, it can be observed that the position of the can can be

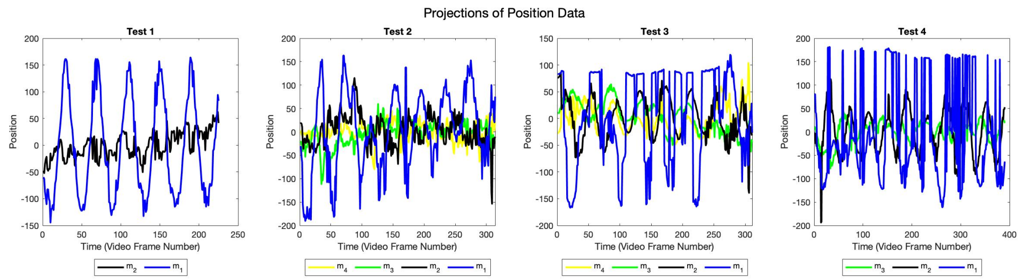


Figure 2: Here are the projections of the position data for each test case. $m_1, \ldots, m_n$ represent the projections for mode $1, \ldots, n$, relating to singular values $1, \ldots, n$.

represented as primarily a sinusoidal motion (relating to the vertical movement of the can). It can also be observed from the figure to the right of it, labeled "Sigma Energies of Test 1," that the first singular value, corresponding to the sinusoidal movement alone, captures about 91% of the energy of the entire phenomenon. The next singular value captures about 7% energy and might represent velocity in a sense, or the offset in motion resulting from the three different camera recordings.

Now examining the Test 2 results, it can be observed that the components appear much noisier than those of Test 1. This makes sense, as this test case had video recordings of the vertical movement that were physically shaking, making it much more difficult to get smooth-looking components. For this case, however, it is still vaguely possible to observe that the first component, relating to about 73% of the total energy, is approximately sinusoidal. This again represents the vertical movement of the can. It is possible that second component again relates to velocity somehow. It may also, in addition to the other components, simply be noise. This plot essentially shows that the addition of noise makes the components appear a lot messier and represent less.

Test 3 introduced horizontal displacement to the situation. The first component, which contained about 68% of the energy of the system, again represented the dominant up-and-down movement of the can. The second component, containing about 13% of the total energy, was also relatively sinusoidal. It seemed to correspond to the newly added horizontal movement. The other components may correspond to noise or another underlying phenomena in the situation.

In Test 4, the can moved vertically, horizontally, and rotated. The first component, containing approximately 76% of the energy of the system, represented the primarily vertical movement of the can. However, some rotation elements may have been added to this principal component, as the plot of this appears much less sinusoidal than for the other cases. The oscillating second component likely represents the horizontal movement of the can. It is likely that the remaining components relate merely to noise or perhaps to another underlying phenomena.

Figure 2 shows the position data projected onto the different modes, corresponding to different singular values, of the system. The plotted data was gathered from the columns of the matrix resulting from the multiplication of the transpose of the **U** matrix and the position data. Using the equation for the SVD, we can observe that

$$\mathbf{U^T X} = \mathbf{\Sigma V}^* \tag{17}$$

Therefore, this plot shows the components of this phenomenon scaled by the $\sigma$ values. This is simply another representation of the trends in the can's movement.

# 5 Summary and Conclusions

In this assignment, the SVD and PCA were used to extract the trends in the position data of a can moving on a string. Four different cases were considered, one for solely vertical motion, one that included a significant amount of noise, one that added in horizontal movement, and one that had both horizontal displacement and rotation. There are many diverse applications for the SVD. This assignment considered just one of them.

# Appendix A MATLAB Functions

- `implay(filename)` uses the Video Viewer app to display the content inside of `filename`.
- `sgtitle(txt)` creates and displays a title specified by the string `txt` for the current subplot grid.
- `[U,S,V] = svd(A,'econ')` returns the reduced singular value decomposition of array `A`.

# Appendix B MATLAB Code

```matlab
clear; close all; clc

%% Load and data for Test 1: Ideal Case
load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')
% implay(vidFrames1_1)
% implay(vidFrames2_1)
% implay(vidFrames3_1)

%% Crop video frame
croppedVidFrames1_1 = vidFrames1_1(:,275:400,:,:);
croppedVidFrames2_1 = vidFrames2_1(:,225:375,:,:);
croppedVidFrames3_1 = vidFrames3_1(225:350,:,:,:);
% implay(croppedVidFrames1_1)
% implay(croppedVidFrames2_1)
% implay(croppedVidFrames3_1)

%% Locate position of the can by tracking the most red component in movie
numFrames1 = size(croppedVidFrames1_1,4);
for j = 1:numFrames1
    X1 = croppedVidFrames1_1(:,:,1,j); % row, col, color, time
    [M,I] = max(X1(:));
    [x1,y1] = ind2sub([size(X1,1), size(X1,2)], I);
    posCam1(j,:) = [x1,y1];
end

numFrames2 = size(croppedVidFrames2_1,4);
for j = 1:numFrames2
    X2 = croppedVidFrames2_1(:,:,1,j); % row, col, color, time
    [M,I] = max(X2(:));
    [x2,y2] = ind2sub([size(X2,1), size(X2,2)], I);
    posCam2(j,:) = [x2,y2];
end

numFrames3 = size(croppedVidFrames3_1,4);
for j = 1:numFrames3
    X3 = croppedVidFrames3_1(:,:,1,j); % row, col, color, time
    [M,I] = max(X3(:));
    [x3,y3] = ind2sub([size(X3,1), size(X3,2)], I);
    posCam3(j,:) = [x3,y3];
end

%% Set the mean of the results equal to zero
```

```
posCam1x = posCam1(:,2) - mean(posCam1(:,2));
posCam1y = posCam1(:,1) - mean(posCam1(:,1));
posCam2x = posCam2(:,2) - mean(posCam2(:,2));
posCam2y = posCam2(:,1) - mean(posCam2(:,1));
posCam3x = posCam3(:,2) - mean(posCam3(:,2));
posCam3y = posCam3(:,1) - mean(posCam3(:,1));

%% Plot results
subplot(1,3,1)
plot(posCam1x)
hold on
plot(posCam1y)
title('Camera 1')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('Horizontal Movement','Vertical Movement','Location','southoutside')

subplot(1,3,2)
plot(posCam2x)
hold on
plot(posCam2y)
title('Camera 2')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('Horizontal Movement','Vertical Movement','Location','southoutside')

subplot(1,3,3)
plot(posCam3y)
hold on
plot(posCam3x)
title('Camera 3')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('Horizontal Movement','Vertical Movement','Location','southoutside')

sgtitle('Position Data for Test 1: Ideal Case')

%% Trim data to line up peaks
tPosCam1x = posCam1x(2:end);
tPosCam1y = posCam1y(2:end);

tPosCam2x = posCam2x(11:length(tPosCam1x)+10);
tPosCam2y = posCam2y(11:length(tPosCam1x)+10);

tPosCam3x = posCam3x(1:length(tPosCam1x));
tPosCam3y = posCam3y(1:length(tPosCam1x));

%% Perform PCA and calculate energies
test1Mat = [tPosCam1x tPosCam1y tPosCam2x tPosCam2y tPosCam3x tPosCam3y];
test1Mat = test1Mat'; % let data vectors be rows of test1Mat
[U1,S1,V1] = svd(test1Mat,'econ');
sig1 = diag(S1);
energies1 = sig1.^2/sum(sig1.^2); % will be plotted later

```

```
%% Repeat for Test 2: Noisy Case

%% Load data for Test 2: Noisy Case
load('cam1_2.mat')
load('cam2_2.mat')
load('cam3_2.mat')

%% Crop video frame
croppedVidFrames1_2 = vidFrames1_2(:,275:500,:,:);
croppedVidFrames2_2 = vidFrames2_2(:,175:450,:,:);
croppedVidFrames3_2 = vidFrames3_2(175:350,:,:,:);

%% Locate position of the can by tracking the most red component in movie
numFrames1 = size(croppedVidFrames1_2,4);
for j = 1:numFrames1
    X1 = croppedVidFrames1_2(:,:,1,j); % row, col, color, time
    [M,I] = max(X1(:));
    [x1,y1] = ind2sub([size(X1,1), size(X1,2)], I);
    posCam1(j,:) = [x1,y1];
end

numFrames2 = size(croppedVidFrames2_2,4);
for j = 1:numFrames2
    X2 = croppedVidFrames2_2(:,:,1,j); % row, col, color, time
    [M,I] = max(X2(:));
    [x2,y2] = ind2sub([size(X2,1), size(X2,2)], I);
    posCam2(j,:) = [x2,y2];
end

numFrames3 = size(croppedVidFrames3_2,4);
for j = 1:numFrames3
    X3 = croppedVidFrames3_2(:,:,1,j); % row, col, color, time
    [M,I] = max(X3(:));
    [x3,y3] = ind2sub([size(X3,1), size(X3,2)], I);
    posCam3(j,:) = [x3,y3];
end

%% Set the mean of the results equal to zero
posCam1x = posCam1(:,2) - mean(posCam1(:,2));
posCam1y = posCam1(:,1) - mean(posCam1(:,1));
posCam2x = posCam2(:,2) - mean(posCam2(:,2));
posCam2y = posCam2(:,1) - mean(posCam2(:,1));
posCam3x = posCam3(:,2) - mean(posCam3(:,2));
posCam3y = posCam3(:,1) - mean(posCam3(:,1));

%% Plot results
figure
subplot(1,3,1)
plot(posCam1x)
hold on
plot(posCam1y)
title('Camera 1')
xlabel('Time (Video Frame Number)')
ylabel('Position')
```

```
legend('Horizontal Movement','Vertical Movement','Location','southoutside')

subplot(1,3,2)
plot(posCam2x)
hold on
plot(posCam2y)
title('Camera 2')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('Horizontal Movement','Vertical Movement','Location','southoutside')

subplot(1,3,3)
plot(posCam3y)
hold on
plot(posCam3x)
title('Camera 3')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('Horizontal Movement','Vertical Movement','Location','southoutside')

sgtitle('Position Data for Test 2: Noisy Case')

%% Trim data
tPosCam1x = posCam1x(1:end);
tPosCam1y = posCam1y(1:end);

tPosCam2x = posCam2x(26:length(tPosCam1x)+25);
tPosCam2y = posCam2y(26:length(tPosCam1x)+25);

tPosCam3x = posCam3x(7:length(tPosCam1x)+6);
tPosCam3y = posCam3y(7:length(tPosCam1x)+6);

%% Perform PCA and calculate energies
test2Mat = [tPosCam1x tPosCam1y tPosCam2x tPosCam2y tPosCam3x tPosCam3y];
test2Mat = test2Mat'; % let data vectors be rows of test2Mat
[U2,S2,V2] = svd(test2Mat,'econ');
sig2 = diag(S2);
energies2 = sig2.^2/sum(sig2.^2);

%% Repeat for Test 3: Horizontal Displacement

%% Load data for Test 3: Horizontal Displacement
load('cam1_3.mat')
load('cam2_3.mat')
load('cam3_3.mat')

%% Crop video frame
croppedVidFrames1_3 = vidFrames1_3(:,275:400,:,:);
croppedVidFrames2_3 = vidFrames2_3(:,175:450,:,:);
croppedVidFrames3_3 = vidFrames3_3(150:350,:,:,:);

%% Locate position of the can by tracking the most red component in movie
numFrames1 = size(croppedVidFrames1_3,4);
for j = 1:numFrames1
```

```
    X1 = croppedVidFrames1_3(:,:,1,j); % row, col, color, time
    [M,I] = max(X1(:));
    [x1,y1] = ind2sub([size(X1,1), size(X1,2)], I);
    posCam1(j,:) = [x1,y1];
end

numFrames2 = size(croppedVidFrames2_3,4);
for j = 1:numFrames2
    X2 = croppedVidFrames2_3(:,:,1,j); % row, col, color, time
    [M,I] = max(X2(:));
    [x2,y2] = ind2sub([size(X2,1), size(X2,2)], I);
    posCam2(j,:) = [x2,y2];
end

numFrames3 = size(croppedVidFrames3_3,4);
for j = 1:numFrames3
    X3 = croppedVidFrames3_3(:,:,1,j); % row, col, color, time
    [M,I] = max(X3(:));
    [x3,y3] = ind2sub([size(X3,1), size(X3,2)], I);
    posCam3(j,:) = [x3,y3];
end

%% Set the mean of the results equal to zero
posCam1x = posCam1(:,2) - mean(posCam1(:,2));
posCam1y = posCam1(:,1) - mean(posCam1(:,1));
posCam2x = posCam2(:,2) - mean(posCam2(:,2));
posCam2y = posCam2(:,1) - mean(posCam2(:,1));
posCam3x = posCam3(:,2) - mean(posCam3(:,2));
posCam3y = posCam3(:,1) - mean(posCam3(:,1));

%% Plot results
figure
subplot(1,3,1)
plot(posCam1x)
hold on
plot(posCam1y)
title('Camera 1')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('Horizontal Movement','Vertical Movement','Location','southoutside')

subplot(1,3,2)
plot(posCam2x)
hold on
plot(posCam2y)
title('Camera 2')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('Horizontal Movement','Vertical Movement','Location','southoutside')

subplot(1,3,3)
plot(posCam3y)
hold on
plot(posCam3x)
```

```
title('Camera 3')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('Horizontal Movement','Vertical Movement','Location','southoutside')

sgtitle('Position Data for Test 3: Horizontal Displacement')

%% Trim data
tPosCam1x = posCam1x(4:end);
tPosCam1y = posCam1y(4:end);

tPosCam2x = posCam2x(24:length(tPosCam1x)+23);
tPosCam2y = posCam2y(24:length(tPosCam1x)+23);

tPosCam3x = posCam3x(1:length(tPosCam1x));
tPosCam3y = posCam3y(1:length(tPosCam1x));

%% Perform PCA and calculate energies
test3Mat = [tPosCam1x tPosCam1y tPosCam2x tPosCam2y tPosCam3x tPosCam3y];
test3Mat = test3Mat'; % let data vectors be rows of test3Mat
[U3,S3,V3] = svd(test3Mat,'econ');
sig3 = diag(S3);
energies3 = sig3.^2/sum(sig3.^2);

%% Repeat for Test 4: Horizontal Displacement and Rotation

%% Load data for Test 4: Horizontal Displacement and Rotation
load('cam1_4.mat')
load('cam2_4.mat')
load('cam3_4.mat')

%% Crop video frame
croppedVidFrames1_4 = vidFrames1_4(:,300:475,:,:);
croppedVidFrames2_4 = vidFrames2_4(:,175:450,:,:);
croppedVidFrames3_4 = vidFrames3_4(125:350,:,:,:);

%% Locate position of the can by tracking the most red component in movie
numFrames1 = size(croppedVidFrames1_4,4);
for j = 1:numFrames1
    X1 = croppedVidFrames1_4(:,:,1,j); % row, col, color, time
    [M,I] = max(X1(:));
    [x1,y1] = ind2sub([size(X1,1), size(X1,2)], I);
    posCam1(j,:) = [x1,y1];
end

numFrames2 = size(croppedVidFrames2_4,4);
for j = 1:numFrames2
    X2 = croppedVidFrames2_4(:,:,1,j); % row, col, color, time
    [M,I] = max(X2(:));
    [x2,y2] = ind2sub([size(X2,1), size(X2,2)], I);
    posCam2(j,:) = [x2,y2];
end

numFrames3 = size(croppedVidFrames3_4,4);
```

```
for j = 1:numFrames3
    X3 = croppedVidFrames3_4(:,:,1,j); % row, col, color, time
    [M,I] = max(X3(:));
    [x3,y3] = ind2sub([size(X3,1), size(X3,2)], I);
    posCam3(j,:) = [x3,y3];
end

%% Set the mean of the results equal to zero
posCam1x = posCam1(:,2) - mean(posCam1(:,2));
posCam1y = posCam1(:,1) - mean(posCam1(:,1));
posCam2x = posCam2(:,2) - mean(posCam2(:,2));
posCam2y = posCam2(:,1) - mean(posCam2(:,1));
posCam3x = posCam3(:,2) - mean(posCam3(:,2));
posCam3y = posCam3(:,1) - mean(posCam3(:,1));

%% Plot results
figure
subplot(1,3,1)
plot(posCam1x)
hold on
plot(posCam1y)
title('Camera 1')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('Horizontal Movement','Vertical Movement','Location','southoutside')

subplot(1,3,2)
plot(posCam2x)
hold on
plot(posCam2y)
title('Camera 2')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('Horizontal Movement','Vertical Movement','Location','southoutside')

subplot(1,3,3)
plot(posCam3y)
hold on
plot(posCam3x)
title('Camera 3')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('Horizontal Movement','Vertical Movement','Location','southoutside')

sgtitle('Position Data for Test 4: Horizontal Displacement and Rotation')

%% Trim data
tPosCam1x = posCam1x(3:end);
tPosCam1y = posCam1y(3:end);

tPosCam2x = posCam2x(1:length(tPosCam1x));
tPosCam2y = posCam2y(1:length(tPosCam1x));

tPosCam3x = posCam3x(1:length(tPosCam1x));
```

```
tPosCam3y = posCam3y(1:length(tPosCam1x));

%% Perform PCA and calculate energies
test4Mat = [tPosCam1x tPosCam1y tPosCam2x tPosCam2y tPosCam3x tPosCam3y];
test4Mat = test4Mat'; % let data vectors be rows of test4Mat
[U4,S4,V4] = svd(test4Mat,'econ');
sig4 = diag(S4);
energies4 = sig4.^2/sum(sig4.^2);

%% Calculate number of sigmas to reach at least 95% energy
allEnergies = [energies1 energies2 energies3 energies4];
totalEnergies = zeros(4,1); % one for each test
numSigmas = zeros(4,1);
for j = 1:length(numSigmas) % for each test
    for k = 1:length(energies1) % all energy vectors are same length
        curEnergy = totalEnergies(j) + allEnergies(k,j);
        if curEnergy < 0.95
            totalEnergies(j) = curEnergy;
            numSigmas(j) = numSigmas(j) + 1;
        else
            totalEnergies(j) = curEnergy;
            numSigmas(j) = numSigmas(j) + 1;
            break
        end
    end
end

totalEnergies
numSigmas

%% Create plot of columns of V and sigma energies
figure
subplot(4,2,1)
plot(V1(:,2),'k','LineWidth',2)
hold on
plot(V1(:,1),'b','LineWidth',2)
title('Test 1 Components for 95% Energy')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('v_2','v_1','Location','southeast','Orientation','horizontal')

subplot(4,2,2)
plot(energies1,'ko','Linewidth',2)
ylim([0 1])
title('Sigma Energies of Test 1')
xlabel('Singular Values')
ylabel('Fraction of Total Energy')

subplot(4,2,3)
plot(V2(:,4),'y','LineWidth',2)
hold on
plot(V2(:,3),'g','LineWidth',2)
plot(V2(:,2),'k','LineWidth',2)
plot(V2(:,1),'b','LineWidth',2)
```

```
title('Test 2 Components for 95% Energy')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('v_4','v_3','v_2','v_1','Location','southeast','Orientation','
    horizontal')

subplot(4,2,4)
plot(energies2,'ko','Linewidth',2)
ylim([0 1])
title('Sigma Energies of Test 2')
xlabel('Singular Values')
ylabel('Fraction of Total Energy')

subplot(4,2,5)
plot(V3(:,4),'y','LineWidth',2)
hold on
plot(V3(:,3),'g','LineWidth',2)
plot(V3(:,2),'k','LineWidth',2)
plot(V3(:,1),'b','LineWidth',2)
title('Test 3 Components for 95% Energy')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('v_4','v_3','v_2','v_1','Location','southeast','Orientation','
    horizontal')

subplot(4,2,6)
plot(energies3,'ko','Linewidth',2)
ylim([0 1])
title('Sigma Energies of Test 3')
xlabel('Singular Values')
ylabel('Fraction of Total Energy')

subplot(4,2,7)
plot(V4(:,3),'g','LineWidth',2)
hold on
plot(V4(:,2),'k','LineWidth',2)
plot(V4(:,1),'b','LineWidth',2)
title('Test 4 Components for 95% Energy')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('v_3','v_2','v_1','Location','southeast','Orientation','horizontal')

subplot(4,2,8)
plot(energies4,'ko','Linewidth',2)
ylim([0 1])
title('Sigma Energies of Test 4')
xlabel('Singular Values')
ylabel('Fraction of Total Energy')

%% Plot resulting projections of data against time
figure
U12 = U1';
proj1 = U12(1:2,:)*test1Mat;
subplot(1,4,1)
```

```
plot(proj1(2,:),'k','LineWidth',2)
hold on
plot(proj1(1,:),'b','LineWidth',2)
title('Test 1')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('m_2','m_1','Location','southoutside','Orientation','horizontal')

U22 = U2';
proj2 = U22(1:4,:)*test2Mat;
subplot(1,4,2)
plot(proj2(4,:),'y','LineWidth',2)
hold on
plot(proj2(3,:),'g','LineWidth',2)
plot(proj2(2,:),'k','LineWidth',2)
plot(proj2(1,:),'b','LineWidth',2)
title('Test 2')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('m_4','m_3','m_2','m_1','Location','southoutside','Orientation','horizontal')

U32 = U3';
proj3 = U32(1:4,:)*test3Mat;
subplot(1,4,3)
plot(proj3(4,:),'y','LineWidth',2)
hold on
plot(proj3(3,:),'g','LineWidth',2)
plot(proj3(2,:),'k','LineWidth',2)
plot(proj3(1,:),'b','LineWidth',2)
title('Test 3')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('m_4','m_3','m_2','m_1','Location','southoutside','Orientation','horizontal')

U42 = U4';
proj4 = U42(1:3,:)*test4Mat;
subplot(1,4,4)
plot(proj4(3,:),'g','LineWidth',2)
hold on
plot(proj4(2,:),'k','LineWidth',2)
plot(proj4(1,:),'b','LineWidth',2)
title('Test 4')
xlabel('Time (Video Frame Number)')
ylabel('Position')
legend('m_3','m_2','m_1','Location','southoutside','Orientation','horizontal')

sgtitle('Projections of Position Data')
```

Listing 1: This is the code used for extracting the position data, finding the principal components and associated $\sigma$ energies of the position data for the different test cases, and calculating the projections of the data onto the different modes.

# Appendix C Additional Figures

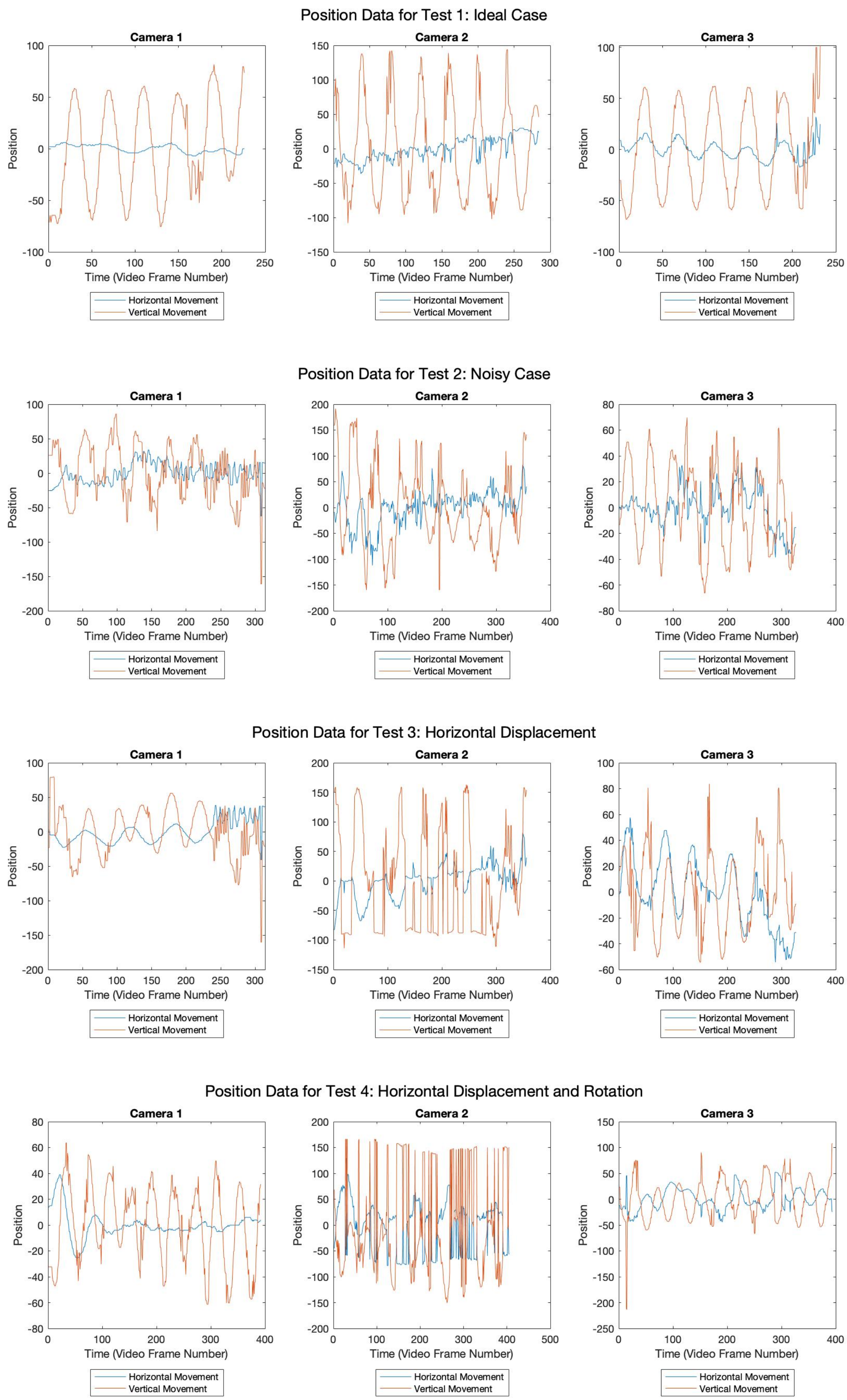


Figure 3: Here are plots of the can's position data for each test case and camera.