AMATH 482 HW 3

Dimensionality Analysis and Low-Rank Approximation

Introduction to Principal Component Analysis

Zachary McNulty
zmcnulty, ID: 1636402

Abstract: Often, the dynamics of a system can be simplified by choosing the most appropriate coordinate system. Not only does this make our system easier to work with computationally, but it helps us better understand its fundamental dynamics. In this paper, we show how Principal Component Analysis (PCA) can be used to reduce the dimensionality of a system, perform low-rank approximations of our system, and make some qualitative inferences about our system. As a case study, we will study the basic spring-mass system.

University of Washington
February 2019

1 Introduction and Overview

When collecting data on a system of interest, it is fairly likely that the underlying dynamics of that system are not completely understood. As a result, the data collected is typically full of redundancies, measuring the same feature of the system from two different perspectives. If we knew the true features of this system, we could choose our coordinate system to reflect them, cutting back on this redundancy. Not only would this reduce the dimensionality of our system and make it computationally easier to work with, but finding these principal features helps us understand the true dynamics of the system. While we can not always determine the true dimensionality of the system with certainty, we will almost always be able to cut out some dimensions that are clearly unimportant. The standard tools for this kind of analysis are the Singular Value Decomposition (SVD) and Principal Component Analysis (PCA). As a case study, we will use PCA to reconstruct the fundamental dynamics of a spring-mass system measured in a heavily redundant way.

2 Theoretical Background

The Singular Value Decomposition is a diagonalization of a given matrix A that focuses on the rotations/reflections and stretching a vector undergoes when transformed by A. Formally, for any matrix $A \in \mathbb{C}^{m \times n}$ there exists a diagonalization of the form:

$$A = U\Sigma V^* \tag{1}$$

where $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ are unitary matrices, $\Sigma \in \mathbb{R}^{m \times n}_{\geq 0}$ is a diagonal matrix, and V^* represents the complex conjugate transpose of V. As U and V are unitary matrices their columns each form an orthonormal basis for their respective space and the matrices have the convenient property that $UU^* = U^*U = I_m$ and $VV^* = V^*V = I_n$. The diagonal entries of Σ are called the **singular values** of A. These singular values and their corresponding columns in U, the **left singular vectors**, and columns in V, the **right singular vectors**, are often arranged within Σ such that $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n \geq 0$. Intuitively, this breaks down the transformation AX into three fundamental transformations: a rotation/reflection V^* within the the domain of the space X, a scaling of the components of the space X by Σ, and another rotation/reflection of the space within the codomain of A. This mathematical tool is used within a data analysis technique called principal component analysis (PCA).

Given some data matrix X whose rows are our different measurements, PCA aims to determine the dimension of X: what is the least number of components that can appropriately summarize my data and what are these components? To answer this, first consider the covaraiance matrix:

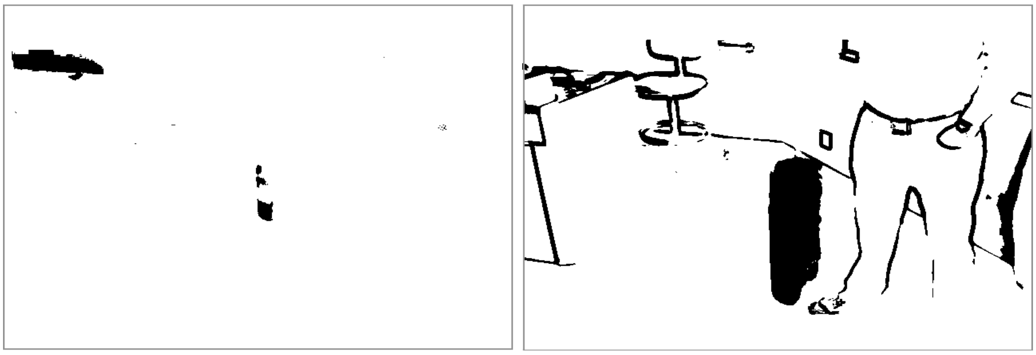
$$C_X = \frac{1}{n-1} XX^T \tag{2}$$

In this matrix, entry (i, j) corresponds to the covariance between row i and row j of X (data measurement i and j). If $i \neq j$, then a high value in entry (i, j) suggests the two data measurements vary in similar ways and are thus likely redundant measurements of the same feature in the data. Consequentially, a low value at (i, j) suggests the two data measurements are fairly independent. If $i = j$, the diagonal of C_X, then entry (i, j) describes the variance in data measurement i. Generally, data measurements with a high variance are assumed to capture important features of the data while those with low variance do not. To see the importance of the SVD, consider the covariance matrix for $Y = U^*X$ where U is from the SVD of X:

$$C_Y = \frac{1}{n-1} YY^T = \frac{1}{n-1} \Sigma^2 \tag{3}$$

As C_Y is diagonal, all its off diagonal entries are zero: projecting the data X onto U^* produces a transformed data set with completely independent measurements, eliminating redundancy in the data. Thus, the columns of U represent the ideal coordinate system to use for our data and the columns of V are the coordinates of our data within this coordinate system. Furthermore, the corresponding singular value σ in the diagonal of Σ ranks the importance of each direction in this coordinate system based on the data's variance along that direction. Note for any given matrix X we can reconstruct it given these coordinates:

$$X = \sum_{i=1}^{min(m,n)} u_j \sigma_j v_j^* \tag{4}$$



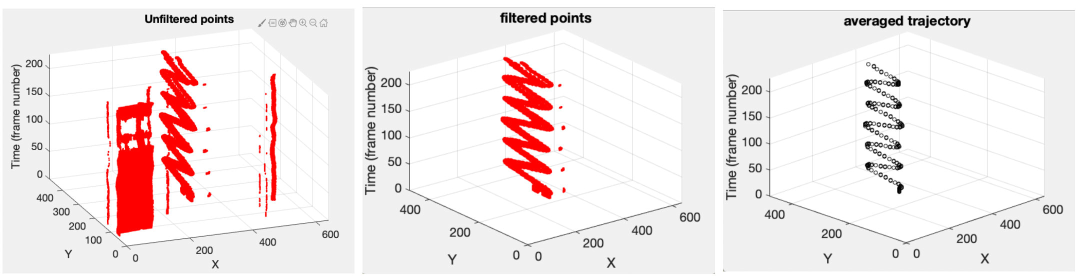
(a) Color Filtered Points (b) Variance Filtered Points

Figure 1: A single frame within the video with points within each frame filtered to find the location of the bucket. Filtering by both color, variance, and position (not shown here) allows for great tracking of the bucket

where u_j, σ_j, v_j are the jth column of U, Σ, V respectively. Since the singular values σ_j are a non-increasing sequence, we see each successive term is increasingly less important to the structure of X. Thus, if the singular values decrease rapidly, we can get a good approximation of X with only the first few terms of this sum. This is a low-rank approximation of the matrix X. The basic idea of PCA is that these first few terms form a good approximation because they represent the fundamental dynamics of the system at hand. These **principal components**, the columns of U with the largest singular values and hence the directions in U along which our data X has the highest variance, best capture the dynamics of our system and the rest is likely noise or less important features.

3 Algorithm Implementation and Development

The first major task is to filter the video data we have for the information we want: the x and y position of the mass (bucket) within each frame of the video. This is performed by the function get_xy_coords(). The white on the bucket and its light provide are some of the brightest areas in the photo so we decided to filter by color. Transforming each frame to black-and-white, we can filter for pixels within the frame that have a high value and are hence bright. We can see in **Figure 1a** that only the bucket and a few other areas of the image have these bright spots. Another way we can filter the frame is by recognizing pixels along the buckets trajectory will have a high variance in their color value. As we see in **Figure 1b** only a few points, mostly edges, have this high variance. Lastly, once we know the general location of the bucket, we can filter points by their x and y coordinates based on a rough approximation of the x and y coordinates the bucket travels through (Appendix B: get_xy_cords() line 1-67). As we can see in **Figure 2a** and **2b**, this combined approach does quite well at eliminating the extraneous pixels. Once we have the filtered pixels in each frame, we average their x and y coordinates to get a single point for each frame, giving the trajectory of the bucket over time. If no point meets the criteria, which rarely occurs, we just use the point from the previous frame (Appendix B: get_xy_cords() line 69-79). **Figure**



(a) Color & Variance filtered (b) Position Filtering (c) Averaged Trajectory

Figure 2: The filtered points plotted against frame number. We can see that position filtering removes much of the noise not removed by color and variance filtering. The averaged trajectory simply averages the x and y coordinates of all the filtered points to get an approximate trajectory.

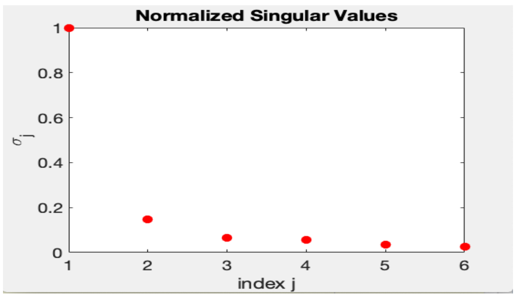
2c shows the results of this averaging. We repeat this process for all three cameras to get a set of three x and y vectors of the bucket over time.

The following steps occur in the my_pca() function. We noticed that some of the videos were not aligned in time. This could cause an issue with PCA as the SVD is sensitive to translations and this offset would alter the phase of the harmonic motion. To rectify this, we simply cut off some of the initial frames of each video so the bucket began at the same position at the start. Furthermore, each video had a different number of frames so we truncated the x and y vectors so they were all the same length without changing the phase (Appendix B: my_pca() line 1-33). We mean subtract each data measurement before placing them in our data matrix X and finding its SVD. From here, we plot the singular values and project our data, X , onto the principal components, the columns of U (Appendix B: my_pca() line 33-57). Once we determine how many singular values we feel are relevant, we can extract the relevant principal components from $Y = U^*X$ and calculate the low-rank approximation using the first few terms of **Equation 4** above corresponding to the relevant singular values. Lastly, we calculate the energy captured by these principal components and plot them as well as the low-rank approximations of each data measurement (Appendix B: my_pca() line 62-157). We repeat this whole process for all four test cases of our data – ideal, noisy, horizontal displacement, and horizontal displacement plus rotation – varying our filtering parameters to find what works best. For example, we found that the variance-based filtering was not as effective in the noisy case so we limited its effect on the filtering.

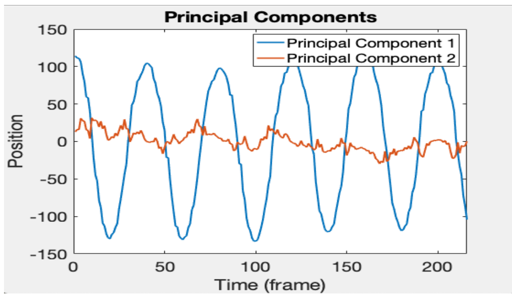
4 Computational Results

4.1 Ideal Case

In the ideal case, the mass is oscillating completely in one dimension and the cameras are held still throughout the recording. As we can see in **Figure 3a**, there is a single dominant singular value. In **Figure 3b**, we see the principal component corresponding to this dominant singular value is the oscillation of the mass we would expect from simple harmonic motion. Furthermore, the second principal component does not seem to be capturing any fundamental behavior of the system and almost appears as zero mean noise. This gives us confidence that our system is truly



(a) Singular Values



(b) Principal Components

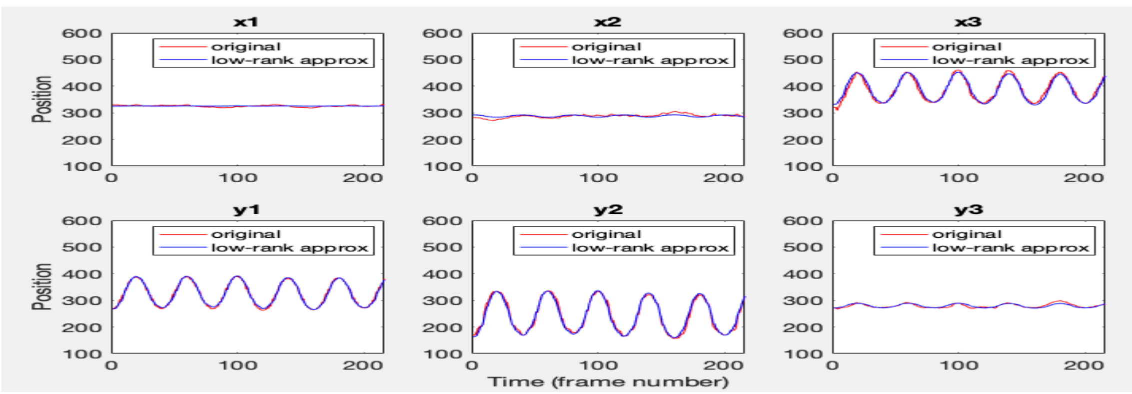


Figure 3: Ideal Case - As we can see, there is clearly one dominant singular value. Its corresponding principal component is clearly a fundamental dynamic of the system. The lower image is the **rank one** approximation of our system.

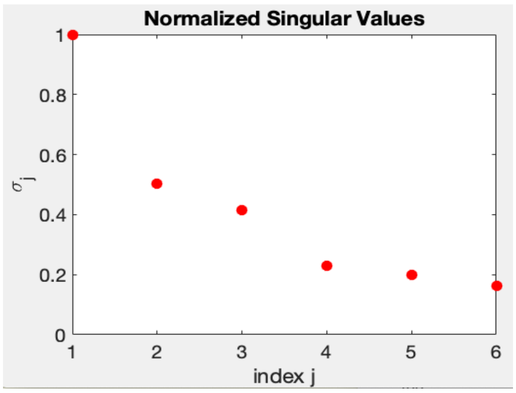
one-dimensional. To support this, we perform the rank one approximation of this system as seen in the lower image of **Figure 3**. We can see this approximation captures the true dynamics incredibly well. With just the one singular value, we capture 75% of the energy of this system. This allows us to confidently claim our system is truly one-dimensional.

4.2 Noisy Case

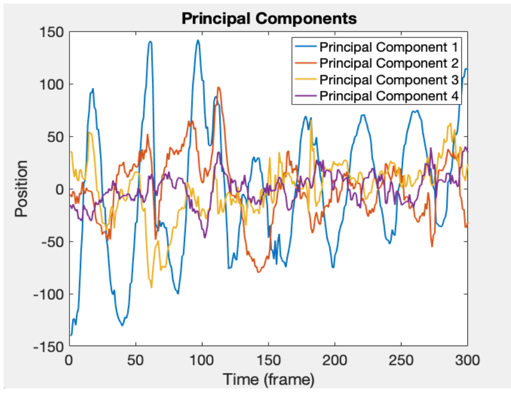
While this form of analysis worked fantastically in the ideal case, introduce some noise and everything becomes a lot more ambiguous. In this case, the noise is introduced in the form of camera shake. Our system is still one-dimensional, but it may be difficult to determine that via PCA. As we can see in **Figure 4a**, there is no longer a sole dominant singular value: σ_2 and σ_3 are arguably relevant as well. Correspondingly, we see in **Figure 4b** that principal component one clearly captures some underlying dynamic of the system, but it is hard to tell whether or not components two and three do as well. Our prior knowledge tells us the system is one-dimensional, but there are facets of the data that are not well-captured by a rank-one approximation. However, as seen in **Figure 4**, the data is almost entirely captured using a rank three approximation and we can see the fourth principal component seems to be nothing more than zero-mean noise. So, while we cannot definitively say our system is one-dimensional as we could before, we can certainly say it is far below six-dimensional. At most it seems there are three important dimensions and at least one of these exhibits what appears to be simple harmonic oscillations (the first principal component). Together, the first three components capture 77% of the energy of the system.

4.3 Horizontal Displacement Case

Now, by giving our mass some side-to-side pendulum motion as well as the simple harmonic oscillation, our system truly is two-dimensional. As we can see in **Figure 5a**, there are clearly two relevant singular values, σ_1 and σ_2. Their corresponding principal components in **Figure 5b**



(a) Singular Values



(b) Principal Components

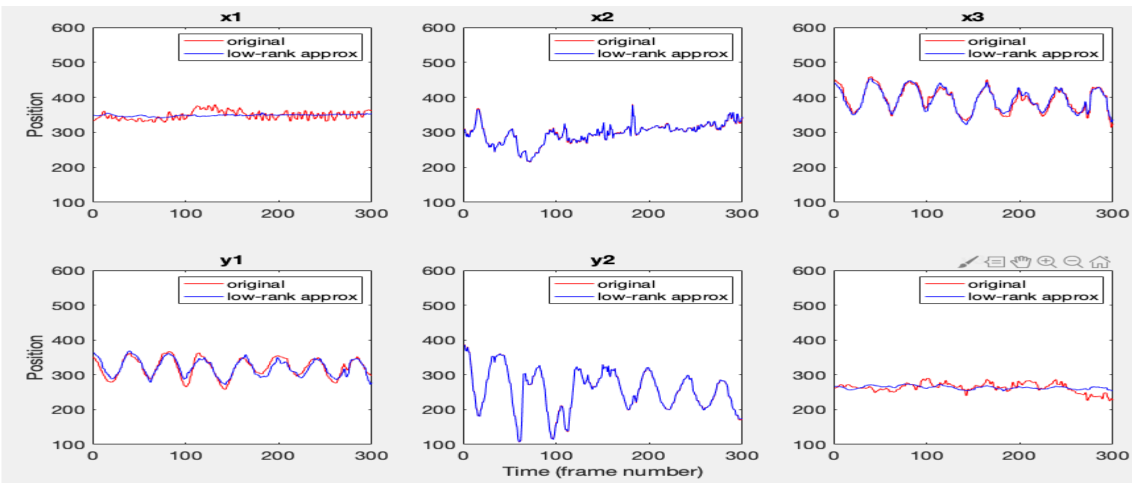


Figure 4: Noisy case - There is no longer a single clear singular values. σ_2 and σ_3 are arguably relevant to our system. The lower image is the **rank three** approximation of our system.

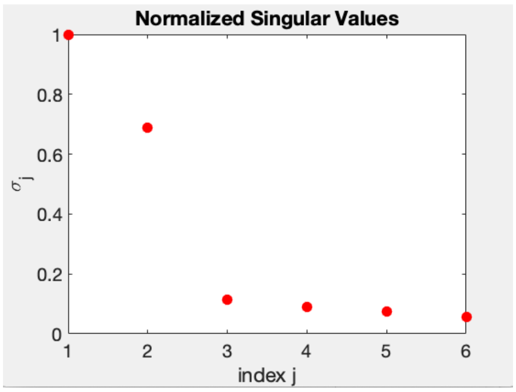
exhibit clear oscillations while further components seem to be nothing more than zero-mean noise, giving us confidence that the system is at most two dimensional. We can see in **Figure 5** that a rank two approximation does a remarkable job of capturing the data. Together, the first two singular values capture 83% of the total energy of the system. Thus, PCA seems to suggest our system is almost completely characterized as a pair of oscillations in two orthogonal directions.

4.4 Horizontal Displacement and Rotation Case

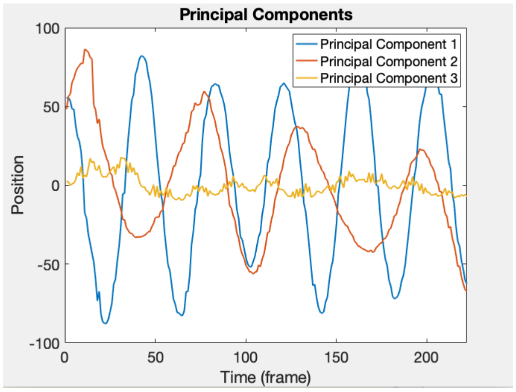
Now not only is the mass oscillating in the z-direction and swinging in the x-y plane, but its also rotating, giving us a three-dimensional system. As we see in **Figure 6a**, there is one clearly relevant singular value σ_1, but σ_2 may also be relevant. The corresponding principal components in **Figure 6b** show a clear oscillation in the first component and what appears to be a damped oscillation in the second component. All other components appear no more than zero-mean noise. While a rank one approximation of the data fails to appropriately capture the data, a rank two approximation as shown in **Figure 6** does an incredible job. This gives us confidence that our system is truly two-dimensional, with a pair of orthogonal oscillations, and that σ_2 is relevant. PCA was not able to reveal any information about the rotation of the can it seems, but this is likely because our image tracking was not sophisticated enough to embed this information into the data. The first two components still capture 75% of the energy of the system.

5 Summary and Conclusions

As we saw in our ideal case, PCA can do a remarkable job of breaking down data into the fundamental components of a system. Unfortunately, this relied on a lot of pre-processing which required some assumptions about our system. In many cases, data is noisy and too little is known about



(a) Singular Values



(b) Principal Components

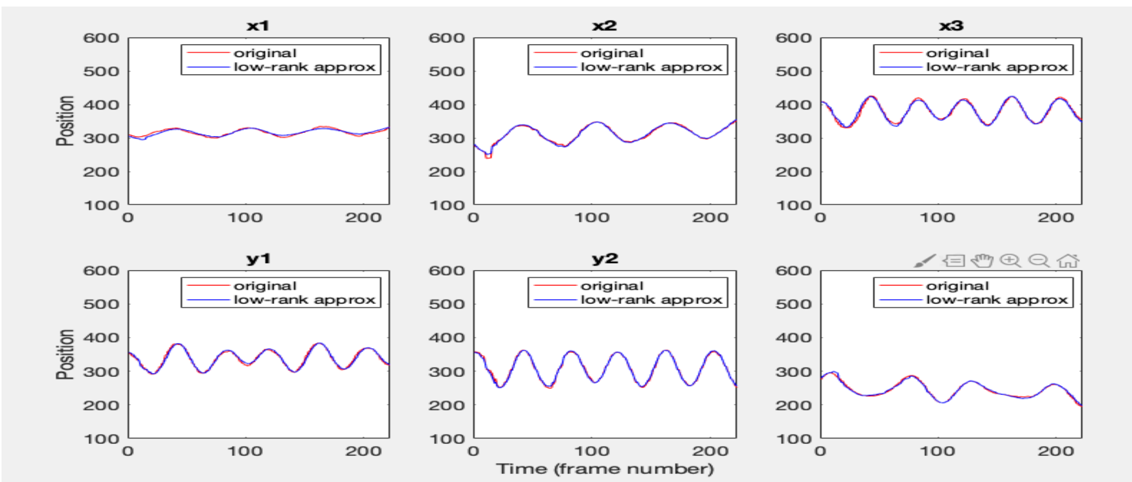
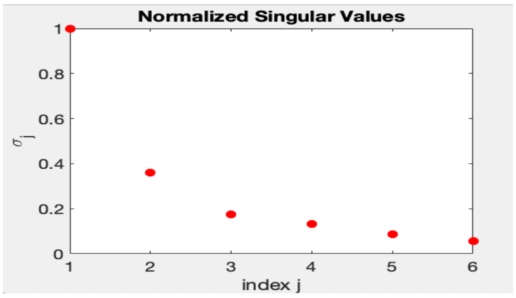
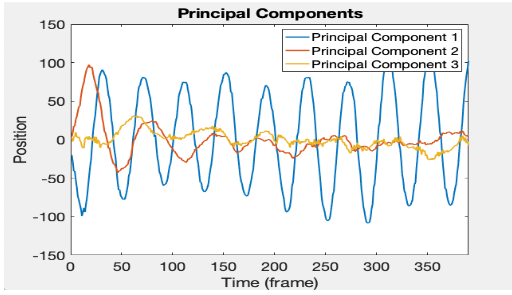


Figure 5: Horizontal case - There are clearly two dominant singular values, σ_1, σ_2. Their corresponding principal components show clear oscillations while the other components seem nothing more than zero-mean noise. The lower image is the **rank two** approximation of our system.



(a) Singular Values



(b) Principal Components

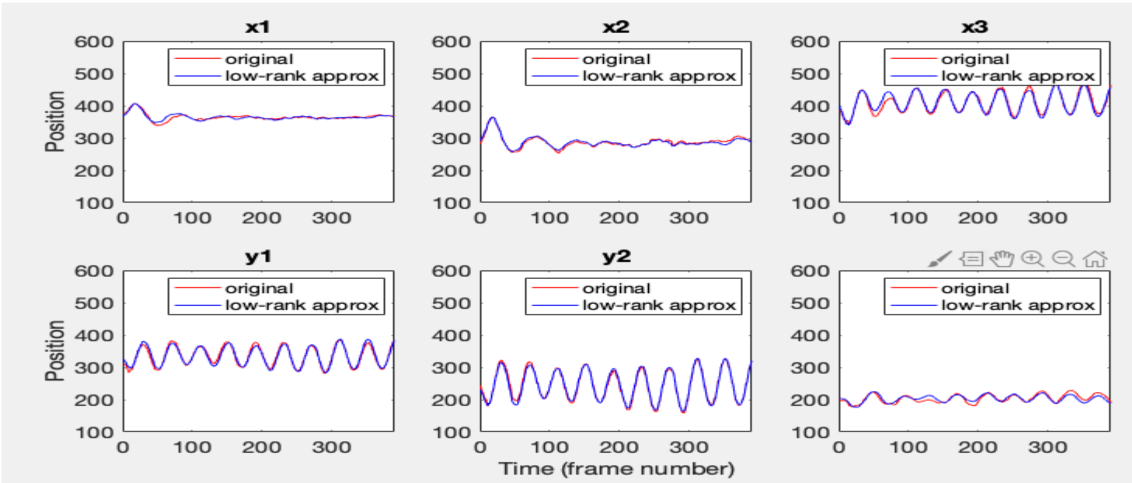


Figure 6: Horizontal and Rotation case - There is a clearly dominant singular value, σ_1, but σ_2 may also be relevant. The lower image is the **rank two** approximation of our system.

the system to make all the necessary pre-processing steps. However, even in the noisy case we saw that PCA could extract some of the fundamental structure of the system. While it could not eliminate all the redundant dimensions with certainty, it still managed to cut the dimensionality in half. Lastly, we saw in the rotation case that PCA could not capture the mass's internal rotation. However, this is likely not a failure in PCA itself but a failure in our data extraction methods: we could not trace the can well enough to embed this rotational information in our data. This is an important lesson that PCA, like many mathematical tools, can only perform as well as the quality of the data it is given.

6 Appendix A

Below is a brief summary of the MATLAB functions I used during this project and their functions.

svd(X): Calculates the Singular Value Decomposition of the given matrix X, returning U, Σ, V.

cov(X): Calculates the covariance matrix C_X of X as described in **Equation 2** above.

diag(X): Returns a vector that contains the diagonal entries of X.

rgb2gray(im): Converts RGB image to grayscale

get_xy_coords(video): A function we wrote that returns the two vectors **x,y** of the averaged x, y coordinates of the paint can in each frame. See code in Appendix B

my_pca(x1,y1, ...): A function we wrote that finds and plots the principal components of the given data as well as the low-rank approximation of the data. See code in Appendix B.

7 Appendix B

7.1 Main Program

```
%% HW 3: Principal Component Analysis (PCA)

% NOTE: convert uint8 to double using double() before processing!
% NOTE: each frame of video should only produce a single timepoint. Taken
%       the mean of the x and y values you find!
% get_xy_coords(video, xrange, yrange, var_scale, max_pixel_val, plots)

%% Part 1: Ideal Case


clear all; close all; clc;

% load data
load('camera_files/cam1_1.mat')
load('camera_files/cam2_1.mat')
load('camera_files/cam3_1.mat')


%% Camera 1 case 1
plots = [0 0 0 0 0 0]; % which plots to show; called in get_xy_coords

close all; clc;

video = vidFrames1_1;
xrange = [300,400];
yrange = [200,450];
var_scale = 1;
max_pixel_val = 240;


[x1_1, y1_1] = get_xy_coords(video, xrange, yrange, var_scale, max_pixel_val, plots);


% Camera 2 case 1
close all; clc;

video = vidFrames2_1;
xrange = [250, 350];
yrange = [100, 375];
var_scale = 1;
max_pixel_val = 240;

[x2_1, y2_1] = get_xy_coords(video, xrange, yrange, var_scale, max_pixel_val, plots);



% Camera 3 case 1
close all; clc;

video = vidFrames3_1;
xrange = [250, 500];
yrange = [225, 325];
var_scale = 1;
max_pixel_val = 240;
```

```
[x3_1, y3_1] = get_xy_coords(video, xrange, yrange, var_scale,
    max_pixel_val, plots);

%% Principal Component Analysis part 1
close all; clc;

rank_approx = 1;

% Some videos are behind the others in time; offset accounts for this
% by aligning the frames
% offset gives which from to start from for video 1,2, and 3
    respectively.
offset = [11, 20, 11];
offset = offset - (min(offset) - 1);
pcs = 2; % number of principal components to plot
yrange = [100, 600];
A = my_pca(rank_approx, pcs, offset, yrange, x1_1, y1_1, x2_1, y2_1,
    x3_1, y3_1);


%% Part 2: Nosiy Case

clear all; close all; clc


load('camera_files/cam1_2.mat')
load('camera_files/cam2_2.mat')
load('camera_files/cam3_2.mat')


%% Camera 1 part 2
close all; clc;

video = vidFrames1_2;
xrange = [300, 400];
yrange = [225, 400];
var_scale = 0.5;
max_pixel_val = 230;
plots = [0 0 0 0 0 0];

[x1_2, y1_2] = get_xy_coords(video, xrange, yrange, var_scale,
    max_pixel_val, plots);


% Camera 2 part 2
close all; clc;

video = vidFrames2_2;
xrange = [175, 450];
yrange = [50, 450];
var_scale = 0.5;
max_pixel_val = 240;
plots = [0 0 0 0 0 0];

[x2_2, y2_2] = get_xy_coords(video, xrange, yrange, var_scale,
    max_pixel_val, plots);



% Camera 3 part 2
```

```matlab
close all; clc;

video = vidFrames3_2;
xrange = [250, 500];
yrange = [225, 300];
var_scale = 0.8;
max_pixel_val = 230;
plots = [0 0 0 0 0 0];

[x3_2, y3_2] = get_xy_coords(video, xrange, yrange, var_scale,
    max_pixel_val, plots);




%% Principal Component Analysis part 2
close all; clc;

rank_approx = 2;
offset = [15, 1, 17];
offset = offset - (min(offset) - 1);
pcs = 4;
yrange = [100, 600];
A = my_pca(rank_approx, pcs, offset, yrange, x1_2, y1_2, x2_2, y2_2,
    x3_2, y3_2);



%% Part 3: Horizontal Displacement

clear all; close all; clc;

load('camera_files/cam1_3.mat')
load('camera_files/cam2_3.mat')
load('camera_files/cam3_3.mat')


%% Camera 1 part 3
close all; clc;

video = vidFrames1_3;
xrange = [250, 400];
yrange = [200, 400];
var_scale = 1;
max_pixel_val = 250;
plots = [0 0 0 0 0 0];

[x1_3, y1_3] = get_xy_coords(video, xrange, yrange, var_scale,
    max_pixel_val, plots);




% Camera 2 part 3
close all; clc;

video = vidFrames2_3;
xrange = [200, 400];
yrange = [175, 400];
var_scale = 1;
max_pixel_val = 240;
```

```
plots = [0 0 0 0 0 0];

[x2_3, y2_3] = get_xy_coords(video, xrange, yrange, var_scale,
    max_pixel_val, plots);




% Camera 3 part 3
close all; clc;

video = vidFrames3_3;
xrange = [250, 450];
yrange = [175, 325];
var_scale = 1;
max_pixel_val = 245;
plots = [0 0 0 0 0 0];

[x3_3, y3_3] = get_xy_coords(video, xrange, yrange, var_scale,
    max_pixel_val, plots);


%% Principal Component Analysis part 3

close all; clc;

rank_approx = 2;

% frame bucket moves to swinger's right
offset = [18 44 9];
offset = offset - (min(offset) - 1);
pcs = 3;
yrange = [100,600];
my_pca(rank_approx, pcs, offset, yrange, x1_3, y1_3, x2_3, y2_3, x3_3,
    y3_3);


%% Part 4: Horizontal Displacement AND Rotation
clear all; close all; clc;

load('camera_files/cam1_4.mat')
load('camera_files/cam2_4.mat')
load('camera_files/cam3_4.mat')

%% Camera 1 part 4

close all; clc;

video = vidFrames1_4;
xrange = [300, 450];
yrange = [225, 400];
var_scale = 1;
max_pixel_val = 245;
plots = [0 0 0 0 0 1];

[x1_4, y1_4] = get_xy_coords(video, xrange, yrange, var_scale,
    max_pixel_val, plots);



%% Camera 2 part 4
```

```matlab
close all; clc;

video = vidFrames2_4;
xrange = [210, 400];
yrange = [100, 350];
var_scale = 1;
max_pixel_val = 250;
plots = [0 0 0 0 0 1];

[x2_4, y2_4] = get_xy_coords(video, xrange, yrange, var_scale,
    max_pixel_val, plots);


%% Camera 3 part 4
close all; clc;

video = vidFrames3_4;
xrange = [300, 500];
yrange = [175, 250];
var_scale = 0.7;
max_pixel_val = 235;
plots = [0 0 0 0 0 1];

[x3_4, y3_4] = get_xy_coords(video, xrange, yrange, var_scale,
    max_pixel_val, plots);



%% Principal Component Analysis part 4
close all; clc;

rank_approx = 2;
offset = [11, 17, 9];
offset = offset - (min(offset) - 1);
pcs = 3;
yrange = [100,600];
my_pca(rank_approx, pcs, offset, yrange, x1_4, y1_4, x2_4, y2_4, x3_4,
    y3_4);
```

7.2 Helper Methods

7.2.1 get_xy_coords()

```
function [all_x ,all_y ] = get_xy_coords(video , xrange , yrange , var_scale
    , max_pixel_val , plots)
% given the video image "video", find the xy coordinates in each frame
% where the bucket is located (i.e. tracks the bucket over time)

% var_scale: controls the variance filter; looks for points with high
% variance and have variance > var_scale * mean_variance

% max_pixel_val: filters the pixels by color, looking for pixels with
%                grayscale color above this value.

% xrange & yrange: describe where to look for the bucket it in
    pixelspace
%                  chosen manually after viewing unfiltered points

% plots: logical vector for which describing which figures to show



% Video loading taken from page 120 of class notes with slight
% modifications

numFrames = size(video, 4);
all_x = [];
all_y = [];

for k = 1 : numFrames
    mov(k).cdata = video(:,:,:,k);
    mov(k).colormap = [];
end

all_Xg = zeros(480,640, numFrames);

% convert video to grayscale
for j=1:numFrames
    X=frame2im(mov(j));
    Xg = rgb2gray(X);
    all_Xg(:,:, j) = Xg;
end

% calculate variances of each pixel over time and the mean variance
% among all pixels
all_variances = var(all_Xg, [], 3);
mean_var = mean(all_variances, 'all');


for j=1:numFrames

    D = uint8(all_Xg(:,:,j));

    % filter pixels by color (in this frame) and by their variance (
        over
    % course of the entire film). Filtering by variance helps eliminate
    % stationary bright points in the background of the film.
    % points that meet the given criteria

    points = logical((all_variances > var_scale*mean_var) .* (D >=
```

```
        max_pixel_val));
    D(points) = 0;
    D(~points) = 255;

    [y_vals, x_vals] = find(points == 1);

    % indices of points within yrange
    filtered_y = (y_vals >= yrange(1)) .* (y_vals <= yrange(2));

    % indices of points within xrange
    filtered_x = (x_vals >= xrange(1)) .* (x_vals <= xrange(2));

    % only take points that are both within yrange AND xrange
    filtered_points = [x_vals(logical(filtered_y.*filtered_x)),  y_vals
        (logical(filtered_x.*filtered_y))];

    % If you cannot find any points, just use last points location.
    if isempty(filtered_points)
        ave_x = all_x(j-1);
        ave_y = all_y(j-1);
    else
        ave_x = mean(filtered_points(:, 1));
        ave_y = mean(filtered_points(:, 2));
    end

    all_x = [all_x, ave_x];
    all_y = [all_y, ave_y];


    % X points of interest vs time

    if plots(1) == 1
        figure(1)
        subplot(211)
        title("X versus time")
        plot(j*ones(1, length(x_vals)), x_vals ,'r.');
        xlim([0, numFrames])
        ylim([0, size(points, 2)])
        hold on

        % Y points of interest vs time
        subplot(212)
        title("Y versus time")
        plot(j*ones(1, length(y_vals)), y_vals ,'r.');
        xlim([0, numFrames])
        ylim([0, size(points, 1)])
        hold on
    end

    if plots(2) == 1
        figure(2)
        imshow(D);
    end

    if plots(3) == 1
        % All points that meet target conditions as a function of time
        figure(3)
        plot3(x_vals, y_vals, j*ones(1,length(y_vals)), 'r.'), grid on;
        xlabel("X")
        ylabel("Y")
        title("Unfiltered points")
```

```
        zlabel("Time (frame number)")
        xlim([0, 640]);
        ylim([0, 480]);
        zlim([0, numFrames]);
        hold on;
        set(gca, 'fontsize', 20);
    end

    if plots(4) == 1
        % filtered paint can trajectory in 3D
        figure(4)
        plot3(filtered_points(:,1), filtered_points(:,2), j*ones(1, length(filtered_points)), 'r.'), grid on;
        hold on
        % also plot "averaged" point
        %plot3(ave_x, ave_y, j, 'ko')
        xlabel("X")
        ylabel("Y")
        title("filtered points")
        zlabel("Time (frame number)")
        xlim([0, 640]);
        ylim([0, 480]);
        zlim([0, numFrames]);
        hold on;
        set(gca, 'fontsize', 20);
    end

    if plots(5) == 1
        %averaged trajectory in 3D
        figure(5)
        plot3(ave_x, ave_y, j, 'ko'), grid on;
        xlabel("X")
        ylabel("Y")
        title("averaged trajectory")
        zlabel("Time (frame number)")
        xlim([0, 640]);
        ylim([0, 480]);
        zlim([0, numFrames]);
        hold on;
        set(gca, 'fontsize', 20);

    end

    if plots(6) == 1
        figure(6)
        all_Xg(round(ave_y)-5:round(ave_y)+5, round(ave_x)-5:round(ave_x)+5, j) = 0;
        imshow(uint8(all_Xg(:,:, j)))
        text(50,100, strcat("Frame Number: ", num2str(j)), 'fontsize', 20, 'BackgroundColor', 'white')
        %pause(0.5)
    end


end

end
```

7.2.2 my_pca()

```matlab
function A = my_pca(rank_approx, pcs, offset, yrange, varargin)

% rank_approx: What dimension of rank-approximation we want to perform.
% i.e. how many of the singular values are relevant?
% pcs = number of principal components to plot
% yrange = for plotting; y limits on low rank approximations
% varargin = time series data; list of vectors x1, y1, x2, etc...

x1 = varargin{1};
y1 = varargin{2};
x1 = x1(offset(1):end);
y1 = y1(offset(1):end);

x2 = varargin{3};
y2 = varargin{4};
x2 = x2(offset(2):end);
y2 = y2(offset(2):end);

x3 = varargin{5};
y3 = varargin{6};
x3 = x3(offset(3):end);
y3 = y3(offset(3):end);

% add all time measurments to a matrix with time varying across columns
% and position measurements as rows. i.e.
% [x1(1) x2(2) ....;
%   y1(1) y2(2) ....]
% This makes U in the SVD describe the principal directions in space and
% V the principal directions in time??

% position vectors are not all the same length; find the min length and
% use that many points instead.
n = min([length(x1); length(x2); length(x3)]);
X = [x1(1:n); y1(1:n); x2(1:n) ; y2(1:n); x3(1:n); y3(1:n)];
means = mean(X.').';


% demean data
X = X - means;


[u, s, v] = svd(X);

% By diagonalizing our covariance matrix, we can generate some
% completely independent component (see pg 117 of AMATH 582/482 notes)

Y = u'*X; % note we want ' not .' as here we want complex conjugate U*
principal_components = Y(1:pcs, :);

% plot the normalized sigma values
singular_values = diag(s);
plot(singular_values ./ max(singular_values), 'r.', 'markersize', 40);
xticks(1:6)
title("Normalized Singular Values")
ylabel('\sigma_j')
xlabel('index j')
set(gca, 'fontsize', 20);

```

```
% Reconstructing X using a low-rank approximation
A = zeros(size(X));

for j = 1:rank_approx
    A = A + singular_values(j).*u(:, j)*v(:, j)'; % transpose is our *
end

% Remean the data!
A = A + means;


% energy captured?
energy = sum(singular_values(1:rank_approx)) / sum(singular_values)


figure(7)
subplot(231)
plot(x1, 'r'), hold on;
ylim(yrange)
xlim([0, n])
title('x1')
plot(A(1,:), 'b')
legend({'original', 'low-rank approx'})
set(gca, 'fontsize', 15);
ylabel('Position')


subplot(234)
plot(y1, 'r'), hold on
ylim(yrange)
xlim([0, n])
title('y1')
plot(A(2,:), 'b')
legend({'original', 'low-rank approx'})
set(gca, 'fontsize', 15);
ylabel('Position')


subplot(232)
plot(x2, 'r'), hold on;
ylim(yrange)
xlim([0, n])
title('x2')
plot(A(3,:), 'b')
legend({'original', 'low-rank approx'})
set(gca, 'fontsize', 15);


subplot(235)
plot(y2, 'r'), hold on;
ylim(yrange)
xlim([0, n])
title('y2')
plot(A(4,:), 'b')
legend({'original', 'low-rank approx'})
xlabel('Time (frame number)')
set(gca, 'fontsize', 15);
```

```
subplot(233)
plot(x3, 'r'), hold on;
ylim(yrange)
xlim([0, n])
title('x3')
plot(A(5,:), 'b')
legend({'original', 'low-rank approx'})
set(gca, 'fontsize', 15);


subplot(236)
plot(y3, 'r'), hold on;
ylim(yrange)
xlim([0, n])
title('y3')
plot(A(6,:), 'b')
legend({'original', 'low-rank approx'})
set(gca, 'fontsize', 15);



% plot the principal_components
figure(8)
names = [];
for k = 1:pcs
    plot(principal_components(k, :), 'linewidth', 2), hold on;
    names = [names strcat("Principal Component ", num2str(k))];
    xlabel('Time (frame)')
    ylabel('Position')

end
    xlim([0, length(principal_components)]);
    title('Principal Components');
    legend(names);
    set(gca, 'fontsize', 20);

end
```