Amath 482 Homework 3

PRINCIPAL COMPONENT ANALYSIS AND LOW-RANK APPROXIMATION

MICHAEL GABALIS

Abstract: This paper aims to demonstrate how Principal Component Analysis (PCA) can be used to reduce the dimensionality of a system, perform low-rank approximations, and make qualitative inferences about the system. Often, the dynamics of a system can be simplified by choosing the most appropriate coordinate system. This makes the system easier to work with computationally and helps us understand the fundamental dynamics. As a case study, we will be looking at a basic spring-mass system with a paint bucket/can as the mass.

University of Washington

February 2021

1 Introduction and Overview

When collecting data on a system of interest, it is common that the underlying dynamics of the system is not completely understood, or there is too much data to make useful inferences. As a result, the data is often full of redundancies, measuring the same feature of the system just from two different perspectives. If we figure out the prominent features of this system, we can choose a coordinate system to reflect them and cut back the redundancy in the data. This will reduce the number of dimensions in our system and make it less computationally difficult, while still figuring out the true dynamics of the system by finding the principal components. The standard tools for doing this analysis are the Singular Value Decomposition (SVD) and the Principal Component Analysis (PCA). As a case study, we will use PCA to reconstruct the fundamental dynamics of a spring-mass system measured with three different cameras. As is clear with this system, we will have lots of redundant data because the cameras will be measuring the same coordinates of the system.

2 Theoretical Background

2.1 Singular Value Decomposition

The Singular Value Decomposition is a diagonalization of a given matrix A that focuses on the rotations or reflections and stretching a vector undergoes when it is transformed by A. For any $A \epsilon \mathrm{C}^{\mathrm{m}\times\mathrm{n}}$ there exists a diagonalization of the form:

$$A = U\Sigma V^* \qquad (1)$$

Where $U \in \mathbb{C}^{m\times n}$ and $V \in \mathbb{C}^{n\times n}$ are unitary matrices, $\Sigma \in \mathbb{R}_{\geq 0}^{m\times n}$ is a diagonal matrix, and V^* is the complex conjugate transpose of V. As U and V are unitary matrices, their columns form an orthonormal basis for their respective space and the matrices have the properties of $UU^* = I_m$ 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 are the left singular vectors and the columns in V are the right singular vectors. These are often arranged within Σ such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. This breaks down the transformation of AX into three different fundamental transformations: a rotation or reflection of V transpose within the domain of X, a scaling of the components the space X by Σ , and another rotation or reflection of the space within the codomain of A.

2.2 Principal Component Analysis

The SVD is used within a data analysis technique called the principal component analysis (PCA). Given some data matrix X whose rows are statistically independent, PCA aims to determine the dimension of X or the least number of components that can sufficiently summarize the data. To find this we must consider the covariance matrix:

$$\mathrm{C}_X = \frac{1}{\mathrm{n}-1}\mathrm{X}\mathrm{X}^{\mathrm{T}} \qquad (2)$$

In this matrix, entry (i,j) corresponds to the covariance between row i and row j of X. If I does not equal to j then a high value in entry (i,j) suggests the two data measurements vary in similar ways and are likely to be

redundant measurements of the same feature in the data. On the other hand, a low value at (i,j) suggests that the two data measurements are largely independent. If i is identical to j, the diagonal of C_x then the entry describes the variance in the data measurement i. Generally, data measurements with a large variance capture important features of the data, but those with low variance do not. To see the important of the SVD, consider the covariance matrix for $Y = U^*X$ where U is from the SVD of X:

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

As $C_y$is the diagonal, all of its diagonal entries are zero; projecting the data X onto U^* produces a transformed data set with completely independent measurements, taking away the 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 system. Also, the corresponding singular value σin the diagonal of Σ ranks the importance of each direction in this system based on the data's variance along that direction. For any given matrix X we can reconstruct it given these coordinates:

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

where u_j , σ_j , v_j^* are the jth column of each SVD component. Since the singular values are a non-increasing sequence, we see that each successive term is increasingly less important to the structure of X. Thus, if the singular values decrease rapidly, we can get an accurate representation of X with only the first few terms of the sum. This is a low-rank approximation of the matrix X. The idea of PCA is that these first few terms form a good approximation to the data set because they represent the fundamental dynamics of the system. These principal components are the columns of U with the largest singular value and the direction U in which our data has the highest variance. This is likely to still capture the dynamics of our system while leaving out the noise or redundancy.

Another valuable tool to look at is the energy of our system. When finding the principal components, we can measure how much energy our low-rank approximation yields. This can measure whether or not we are accurately approximating the data of dimension n with a rank r matrix:

$$\text{energy} = \frac{\sigma_1^2 + \cdots + \sigma_{\mathrm{n}}^2}{\sigma_1^2 + \cdots + \sigma_{\mathrm{r}}^2} \tag{5}$$

3 Algorithm Implementation and Development

3.1 Filtering the Video Data

The first task is to filter the video data we have for the information we are seeking: the x and y coordinates of the mass (bucket in this example) within each frame of the video. This is performed by the function

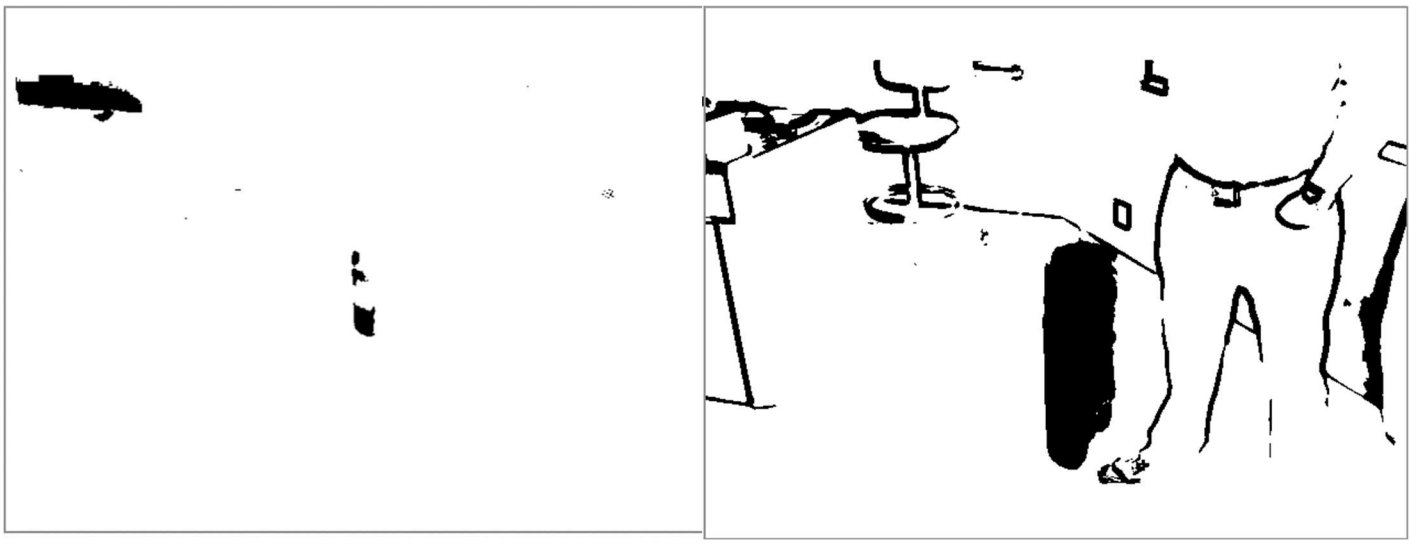

Figure 1: A single frame within the video that is filtered to find the location of the bucket. Filtering by color, variance, and position allows for great tracking of the bucket. (a) On the left is the color filtered points. (b) On the right is the variance filtered points.

get_xy_coords. The white on the bucket and its light provide some of the brightest areas in the photo so we decided to filter it by color. Transforming each frame to bland and white, we can filter for pixels within the frame that have a high value. We can see by **figure 1a,** that only the bucket and a few other areas of the image have bright spots. Another way we can filter the frame is by recognizing pixels along the bucket's trajectory will have a high variance in their color value. As seen in **figure 1b¸** only a few points have this high variance. Once we know the general location of the bucket, we can filter points by their coordinates based on a rough approximation for the path the bucket travels through (Appendix B: get_xy_coords line 1-67). As we can see in **figure 2a** and **figure 2c**, this approach does well at eliminating the extraneous pixels. Once we have the filtered

Pixels in each frame we can average their coordinates to get a single point for each frame, giving the trajectory of the bucket over time (Appendix B: get_xy_coords line 42-50). **Figure 2b,** shows the results of this averaging. We repeat this process for all three cameras to get a set of x and y vectors of the bucket over time.

3.2 Principal Component Analysis

We will now look at my PCA function: my_pca. Some of the videos were not aligned in time, maybe from an issue with PCA as the SVD is sensitive to translations and this offset could alter the phase of the harmonic motion. To counter this, we can cut off some of the initial frames of each video so that the bucket starts in the same position at the start. Additionally, each video had a different number of frames, so we truncated the x and y vectors so that they were all the same length without changing the phase (Appendix B: my_pca line 1-20). Then we subtract the mean of each data measurement before placing them in our data matrix and finding its SVD. From this we can plot the singular values and project our data onto the principal components (the columns of U) (Appendix B: my_pca line 21-38). Once we find how many singular values, we feel are relevant, we can extract the relevant principal components using the SVD and calculate the low-rank approximation using the first few terms of **equation 4**. 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 39- 127). We repeat this entire process for all four test cases of our data, the ideal, noisy, horizontal displacement, and horizontal displacement with rotation.

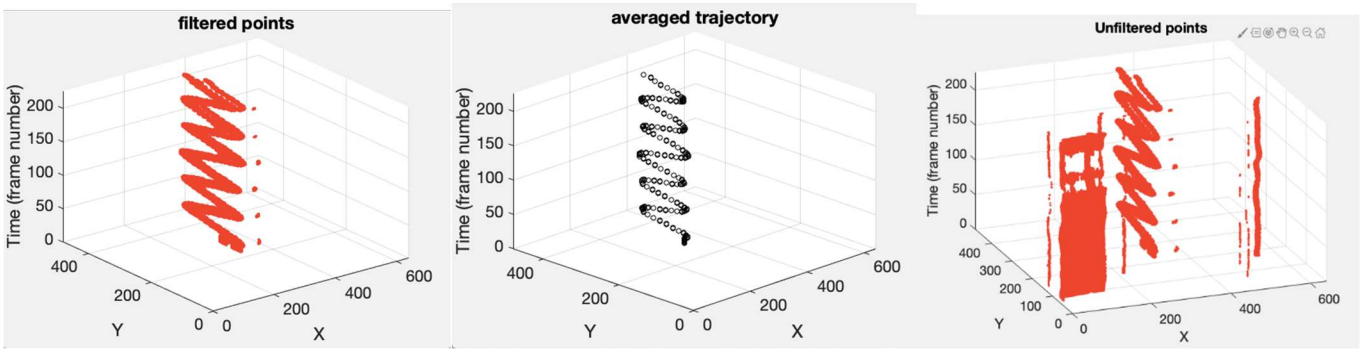

Figure 2: The filtered points plotted against the frame number. The position filtering removes almost all the noise not removed by the color and variance filtering. The averaged trajectory averages the coordinates of all the points. (a) Far left graph shows the position filtering. (b) Middle graph shows the color and variance filtering. (c) Shows the averaged trajectories.

We vary the filtering parameters to find out which work best for each case. As an example, we limited the magnitude of variance-based filtering on the noisy case because it was not as effective.

4 Computational Results

4.1 Ideal Case

In the ideal case, the mass is oscillating is one dimension and the cameras are held still throughout the recording. As we cab see in **figure 3b,** there is a single dominant singular value and as we can see in **figure 3a,** the principal component corresponding to this dominant singular value is the oscillation of the mass we would expect from simple harmonic motion. So, the second principal component does not seem to be capturing any interesting or necessary behavior of our system and seems to be close to zero mean noise. This means we can confidently say that the system is one-dimensional. To support this, we can perform the rank one approximation of this system. We can see this approximation captures the true dynamics of the system incredibly well. With only one singular value, we can capture up to 75% of the energy in the system. Again, allowing us to confidently say that one singular value suffices.

4.2 Noisy Case

While PCA worked very well in the ideal case, when we introduce some noise everything becomes a lot more difficult to analyze. In our case study, the noise is introduced by some sort of camera shaking. Our system still should be one-dimensional, but this may be more difficult to determine by using PCA. As we can see in **figure 4b,** there is no longer a dominant singular value, because it seems that the second and third singular values are also relevant, and the rest never completely go to zero. As we can see in **figure 4a,** principal component one clearly captures some underlying dynamic of the system, but it is difficult to see on whether singular values 1 and 2 do as well. The only thing we can tell is that while principal component 1 seems to give a clear pattern of oscillations, principal component 2 and 3 do not offer any clear patterns. Our prior knowledge tells us that the system is one-dimensional, but there are multiple places in the graph where the data is not captured well by a rank-one approximation. However, in the noisy case I included a graph showing the low-rank approximation across all three cameras in the x and y direction, so that we can more clearly analyze the system. As seen in **figure 4c,** the data is almost completely captured using a rank-three approximation and the fourth principal component seems to be zero mean noise. So, we cannot say that the system is one-dimensional as we could in the ideal case, but we can say that it is at most three-dimensional.

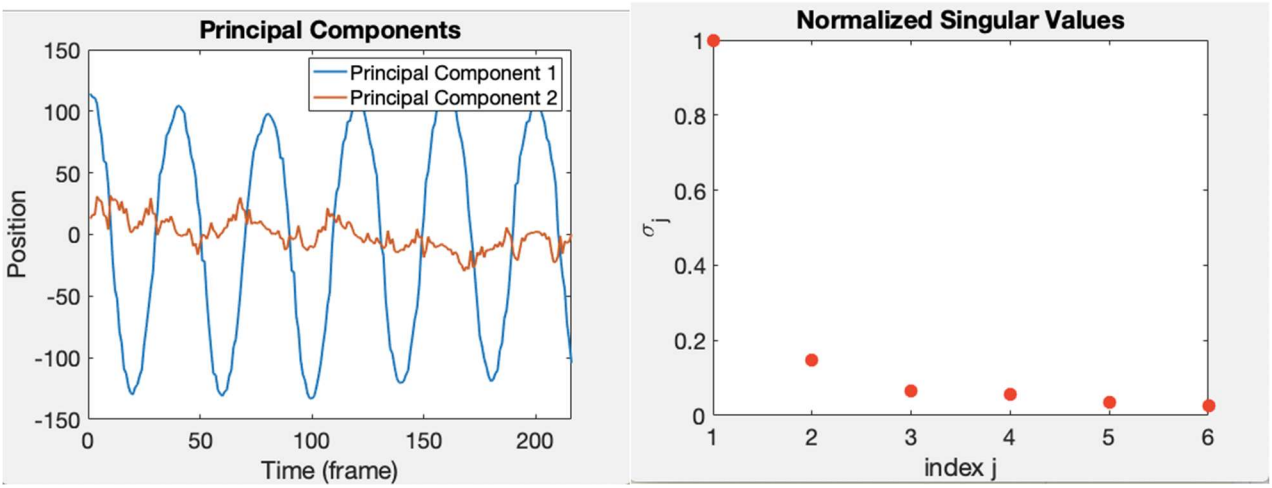

Figure 3: Ideal Case. (a) Left: Principal components. (b) Right: Singular Values. There is clearly only one dominant singular value. Its corresponding principal component is the fundamental dynamic of the system.

A single principal component seems to exhibit the simple harmonic oscillations that we are looking for and we know that the first three components capture 77% of the energy in the system.

4.3 Horizontal Displacement Case

Now we look at the case where the mass is given some horizontal pendulum motion as well as the simple harmonic motion seen in the ideal case.

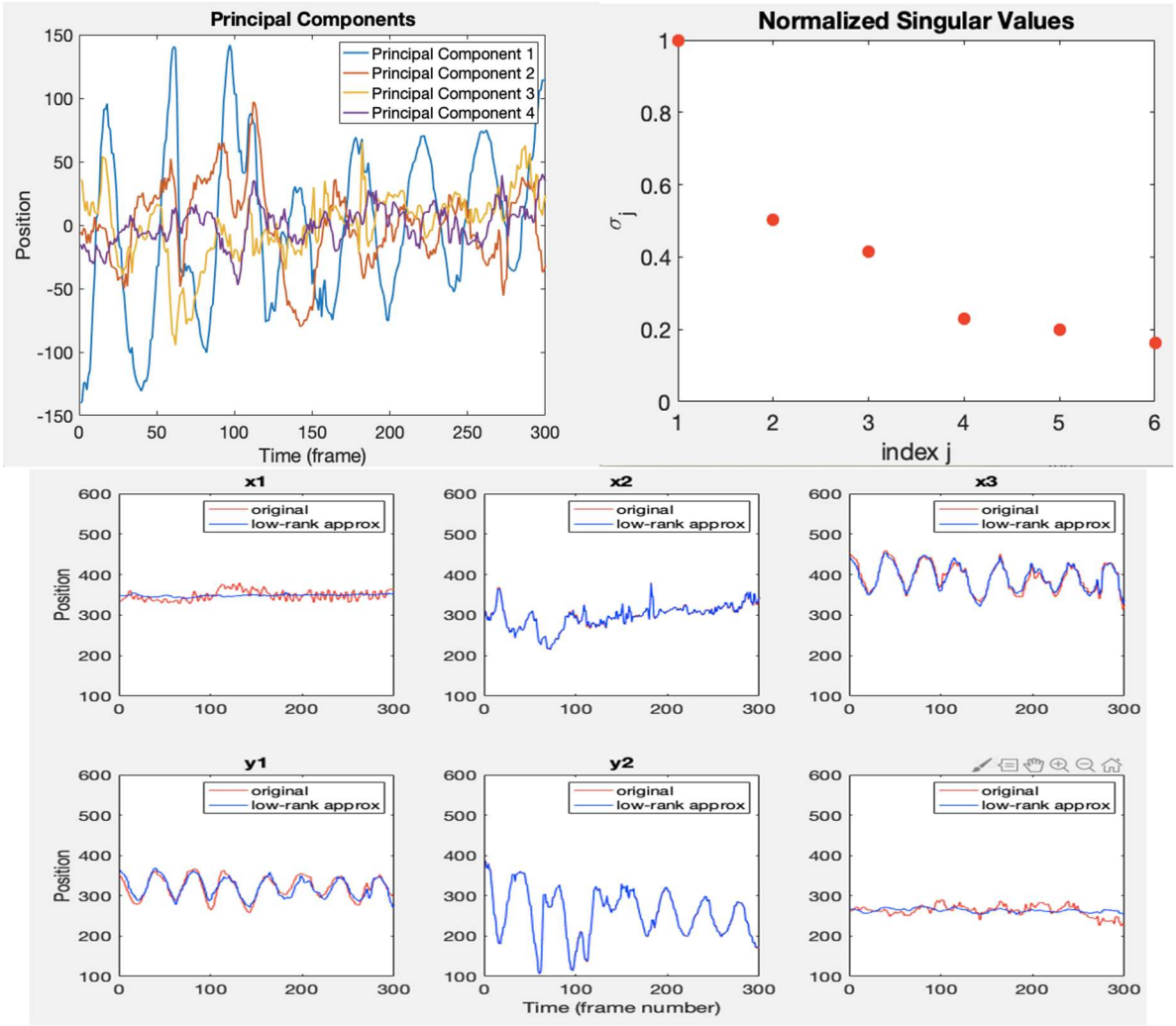

Figure 4: Noisy Case. (a) Top left: Principal Components. (b) Top right: Singular Values. (c): Bottom: Low-rank approximation. As we can see there is now longer a clear singular value in our system.

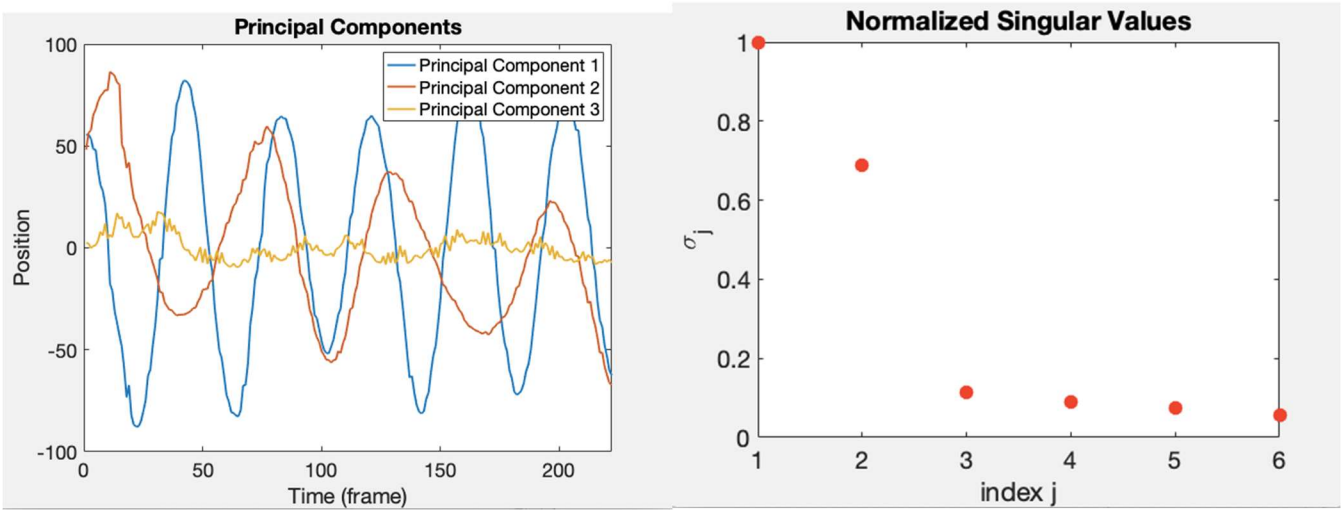

Figure 5: Horizontal Case. (a) Left: Principal components. (b) Right: Singular Values. There is clearly only one dominant singular value. Its corresponding principal component is the fundamental dynamic of the system

This should make our system truly two-dimensional as we will have to account for the horizontal and harmonic motion. As we can see in **figure 5b,** there are clearly two relevant singular values. The other singular values do not contribute much to the system. The corresponding principal components seen in **figure 5a,** show the clear oscillations for the first two principal components while the third component seems to be nothing more than zero mean noise, proving that this system is indeed two-dimensional. A two-rank approximation does a great job as 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. What is also interesting is that it seems like as time goes on the second oscillation for the second principal component seems to decay, also what we know should happen in a damped system.

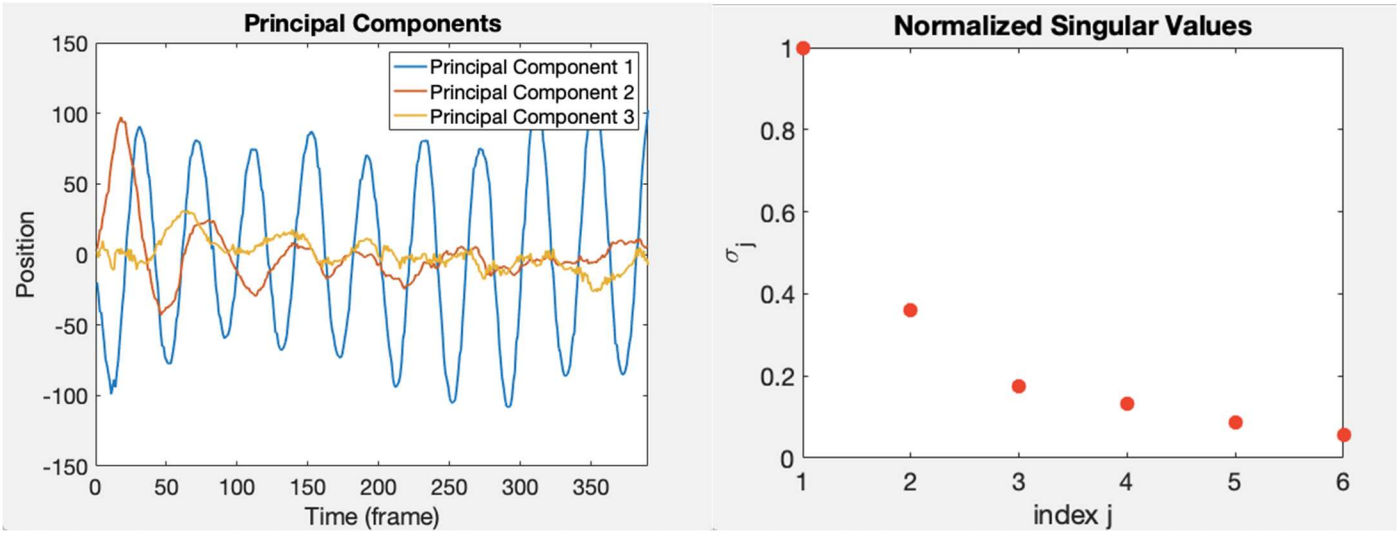

Figure 6: 5: Horizontal and Rotational Case. (a) Left: Principal components. (b) Right: Singular Values. There is clearly only one dominant singular value. Its corresponding principal component is the fundamental dynamic of the system

4.4 Horizontal Displacement with Rotation

Now not only is the mass oscillating with respect to the z-direction and swinging in the x-y plane, but it is also rotating, giving us what should be a three-dimensional system to work with. As we see in **figure 6b,** there is clearly one relevant singular value, but the second and maybe even third singular value is also relevant. The principal components shown in **figure 6a,** shows a clear oscillation in the first component and a damped oscillation in the second component. The third component seems to show only zero-mean noise. While a rank one approximation would fail to appropriately capture the data, a rank two approximation would appropriately capture the significant aspects in the data. This shows that the system is two-dimensional with two different oscillations corresponding the first and second principal component. Unfortunately, PCA was unable to provide any good information about the rotation, but this may be because our camera is not sophisticated enough the capture or embed this information into the data. As a result, the first two components still capture 75% of the energy in the system.

5 Summary and Conclusions

Principal Component Analysis can successfully break down the data into the fundamental components of a system, providing computational speed without removing accuracy. As we saw in our ideal case, PCA was able to create a low-rank approximation that did well at approximating the original position of the bucket with only one Principal Component. The only downside was the amount of pre-processing done that required assumptions about how the system was supposed to perform. In many cases, data is noisy, and little is known about the system to make all the necessary preprocessing steps. That said, even in the noisy case, PCA was able to extract some of the fundamental structure of the system, without sacrificing too much computational power. While unable to eliminate all redundant dimensions with high certainty, we still managed to drastically reduce the dimensionality. PCA was even able to successfully approximate the oscillations in the horizontal displacement case. Both the harmonic motion from the ideal case and the damped oscillation created by a horizontal displacement were able to be approximated with only two principal components. However, in the last case when rotation was added, PCA was unable to capture the mass's internal rotation. This may not be a failure in PCA itself, but a failure in our data extraction methods because we could not accurately trace the bucket well enough to embed this rotational information into the data. It is important to note that PCA, like any other mathematical tool, can only perform as well as the quality of data that is given.

6 Appendix A

Below is the following MATLAB functions I used and a brief description of their functions:

cov(x): Calculates the covariance matrix Cx of x as described in Equation 2 above.

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

get xy coords(video): Returns the two vectors x,y of the averaged x, y coordinates of the paint bucket in each frame. See code in Appendix B

my pca(x1,y1, ...): 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.

rgb2gray(image): Converts an RGB image to grayscale

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

7 APPENDIX B

MATLAB Code

Main Code:

```
clear,clc
%Michael Gabalis
%% HW 3: Principal Component Analysis (PCA)

% NOTE: convert uint8 to double using double() before processing!
% Each frame of video should only produce a single timepoint
%% Part 1: Ideal Case
    % load data
load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')

% Camera 1
plots = [0 0 0 1 0 0]; % which plots to show
close all; clc;
video = vidFrames1_1;
xrange = [300,400];
yrange = [200,400];
var = 1;
max_pval = 240;
[x1_1, y1_1] = get_xy_coords(video, xrange, yrange, var, max_pval, plots);
pause(1)

% Camera 2
close all; clc;

video = vidFrames2_1;
xrange = [250, 350];
yrange = [100, 350];
var = 1;
max_pval = 240;
[x2_1, y2_1] = get_xy_coords(video, xrange, yrange, var, max_pval, plots);

% Camera 3
```

```
close all; clc;

video = vidFrames3_1;
xrange = [250, 500];
yrange = [250, 350];
var = 1;
max_pval = 240;
[x3_1, y3_1] = get_xy_coords(video, xrange, yrange, var, max_pval, plots);

%% Principal Component Analysis part 1
close all; clc;
rank_approx = 1;
% offset accounts by aligning the frames
offset = [11, 20, 11];
offset = offset - (min(offset) - 1);
pc = 2; % number of principal components to plot
yrange = [100, 600];
A = my_pca(rank_approx, pc, offset, yrange, x1_1, y1_1, x2_1, y2_1, x3_1, y3_1);
%% Part 2: Nosiy Case

close all;clc
load('cam1_2.mat')
load('cam2_2.mat')
load('cam3_2.mat')

% Camera 1
close all

video = vidFrames1_2;
xrange = [300, 400];
yrange = [250, 400];
var = 0.5;
max_pval = 230;
plots = [0 0 0 0 0 0];
[x1_2, y1_2] = get_xy_coords(video, xrange, yrange, var, max_pval, plots);

% Camera 2
```

```
close all

video = vidFrames2_2;
xrange = [175, 450];
yrange = [50, 450];
var = 0.5;
max_pval = 240;
plots = [0 0 0 0 0 0];
[x2_2, y2_2] = get_xy_coords(video, xrange, yrange, var, max_pval, plots);
% Camera 3
close all

video = vidFrames3_2;
xrange = [250, 500];
yrange = [225, 300];
var = 0.8;
max_pval = 230;
plots = [0 0 0 0 0 0];
[x3_2, y3_2] = get_xy_coords(video, xrange, yrange, var, max_pval, plots);

%% Principal Component Analysis part 2
close all

rank_approx = 3;
offset = [15, 1, 17];
offset = offset - (min(offset) - 1);
pc = 4;
yrange = [100, 600];
A = my_pca(rank_approx, pc, offset, yrange, x1_2, y1_2, x2_2, y2_2, x3_2, y3_2);

%% Part 3: Horizontal Displacement


close all
load('cam1_3.mat')
load('cam2_3.mat')
load('cam3_3.mat')
```

```
108
109 % Camera 1
110 close all
111
112 video = vidFrames1_3;
113 xrange = [250, 400];
114 yrange = [200, 400];
115 var = 1;
116 max_pval = 250;
117 plots = [0 0 0 0 0 0];
118 [x1_3, y1_3] = get_xy_coords(video, xrange, yrange, var, max_pval, plots);
119
120 % Camera 2
121 close all
122
123 video = vidFrames2_3;
124 xrange = [200, 400];
125 yrange = [175, 400];
126 var = 1;
127 max_pval = 240;
128 plots = [0 0 0 0 0 0];
129 [x2_3, y2_3] = get_xy_coords(video, xrange, yrange, var, max_pval, plots);
130
131 % Camera 3
132 close all
133
134 video = vidFrames3_3;
135 xrange = [250, 450];
136 yrange = [175, 325];
137 var = 1;
138 max_pval = 245;
139 plots = [0 0 0 0 0 0];
140 [x3_3, y3_3] = get_xy_coords(video, xrange, yrange, var, max_pval, plots);
141
142 %% Principal Component Analysis part 3
143 close all; clc;
144
```

```
145 rank_approx = 2;
146  % frame bucket moves
147 offset = [18 44 9];
148 offset = offset - (min(offset) - 1);
149 pc = 3;
150 yrange = [100,600];
151 my_pca(rank_approx, pc, offset, yrange, x1_3, y1_3, x2_3, y2_3, x3_3, y3_3);
152
153 %% Part 4: Horizontal Displacement and Rotation
154 close all; clc;
155
156 load('cam1_4.mat')
157 load('cam2_4.mat')
158 load('cam3_4.mat')
159
160  % Camera 1
161
162 close all; clc;
163 video = vidFrames1_4;
164 xrange = [300, 450];
165 yrange = [225, 400];
166 var = 1;
167 max_pval = 245;
168 plots = [0 0 0 0 0 0];
169 [x1_4, y1_4] = get_xy_coords(video, xrange, yrange, var, max_pval, plots);
170
171  % Camera 2
172 close all;
173
174 video = vidFrames2_4;
175 xrange = [210, 400];
176 yrange = [100, 350];
177 var = 1;
178 max_pval = 250;
179 plots = [0 0 0 0 0 0];
180 [x2_4, y2_4] = get_xy_coords(video, xrange, yrange, var, max_pval, plots);
181
```

```
% Camera 3
close all

video = vidFrames3_4;
xrange = [300, 500];
yrange = [175, 250];
var = 0.7;
max_pval = 235;
plots = [0 0 0 0 0 0];
[x3_4, y3_4] = get_xy_coords(video, xrange, yrange, var, max_pval, plots);

%% Principal Component Analysis part 4
close all

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

Code 2: Get_xy_coords

```
function [ax,ay] = get_xy_coords(video, xrange, yrange, var, max_pval, plots)
% from the video find the xy coordinates where the bucket is located
% var_scale: looks for points with high variance
% max_pval: filters the pixels by color, looking for pixels with
% grayscale color above this value.
% xrange & yrange: viscinity of the bucket in pixelspace chosen manually
%after viewing unfiltered points
% plots: plot ordering

%Video loading
numFrames = size(video, 4);
ax = [];
ay = [];
for k = 1 : numFrames
    mov(k).cdata = video(:,:,:,k);
```

```
16        mov(k).colormap = [];
17  end
18  all_Xg = zeros(480,640,  numFrames);
19
20  % convert  video to  grayscale
21  for  j=1:numFrames
22        X=frame2im(mov(j));
23        Xg = rgb2gray(X);
24        all_Xg(:,:, j) = Xg;
25  end
26  % calculate variances of each pixel over time  and the mean variance
27  % among all pixels
28  variances = var(all_Xg,  [], 3);
29  mean_variance = mean(variances,  'all');
30
31  for  j=1:numFrames
32        D  = uint8(all_Xg(:,:,j));
33        % filter pixels by  color and by their variance
34        points = logical((variances > var*mean_variance) .* (D >=  max_pval));
35        D(points) = 0;
36        D(~points)  = 255;
37        [y, x] = find(points == 1);
38        yfil = (y >= yrange(1)) .* (y <= yrange(2));% indices  of points within yrange
39        xfil = (x >= xrange(1)) .* (x <= xrange(2));% indices  of points within xrange
40        pfil = [x(logical(yfil.*xfil)),    y(logical(xfil.*yfil))];% points within yrange
AND xrange
41        % Use last points location if none in range.
42        if  isempty(pfil)
43              xave =  ax(j-1);
44              yave =  ay(j-1);
45        else
46              xave =  mean(pfil(:, 1));
47              yave =  mean(pfil(:, 2));
48        end
49        ax = [ax, xave];
50        ay = [ay, yave];
```

```
51      %X points of interest vs time
52      if plots(1) == 1
53          figure(1)
54          subplot(211)
55          plot(j*ones(1, length(x)), x ,'r.');
56          title("X versus time")
57          xlim([0, numFrames])
58          ylim([0, size(points, 2)])
59          hold on
60          % Y points of interest vs time
61          subplot(212)
62          plot(j*ones(1, length(y)), y ,'r.');
63          title("Y versus time")
64          xlim([0, numFrames])
65          ylim([0, size(points, 1)])
66          hold on
67      end
68
69      if plots(2) == 1
70          figure(2)
71          imshow(D);
72      end
73
74      if plots(3) == 1
75          % All points that meet target conditions as a function of time
76          figure(3)
77          plot3(x, y, j*ones(1,length(y)), 'r.'), grid on;
78          xlabel("X")
79          ylabel("Y")
80          title("Unfiltered points")
81          zlabel("Time (frame number)")
82          xlim([0, 640]);
83          ylim([0, 480]);
```

```
84              zlim([0, numFrames]);
85              hold on;
86              set(gca, 'fontsize', 20);
87          end
88
89          if plots(4) == 1
90              % filtered paint can trajectory in 3D
91              figure(4)
92              plot3(pfil(:,1), pfil(:,2), j*ones(1,length(pfil)), 'r.'), grid on;
93              hold on
94              % also plot "averaged" point
95              %plot3(ave_x, ave_y, j, 'ko')
96              xlabel("X")
97              ylabel("Y")
98              title("filtered points")
99              zlabel("Time (frame number)")
100             xlim([0, 640]);
101             ylim([0, 480]);

102             zlim([0, numFrames]);
103             hold on;
104             set(gca, 'fontsize', 20);
105         end
106
107         if plots(5) == 1
108             %averaged trajectory in 3D
109             figure(5)
110             plot3(xave, yave, j, 'ko'), grid on;
111             xlabel("X")
112             ylabel("Y")
113             title("averaged trajectory")
114             zlabel("Time (frame number)")
115             xlim([0, 640]);
116             ylim([0, 480]);
117             zlim([0, numFrames]);
118             hold on;
```

```
119             set(gca, 'fontsize', 20);
120         end
121         if plots(6) == 1
122             figure(6)
123             all_Xg(round(yave)-5:round(yave)+5, round(xave)-5:round(xave)+5, j) = 0;
124             imshow(uint8(all_Xg(:,:, j)))
125             text(50,100, strcat("Frame Number: ", num2str(j)), 'fontsize', 20, ↙
'BackgroundColor', 'white')
126         end
127 end
128 end
```

Code 3: My_pca

```
1 function A = my_pca(rank_approx, pc, offset, yrange, varargin)
2 % rank_approx: How many of the singular values are relevant?
3 % pcs = number of principal components to plot
4 % yrange = y limits on low rank approximations
5 % varargin = time series data;
6
      7 x1 = varargin{1};
      8 y1 = varargin{2};
9 x1 = x1(offset(1):end);
10 y1 = y1(offset(1):end);
11
12 x2 = varargin{3};
13 y2 = varargin{4};
14 x2 = x2(offset(2):end);
15 y2 = y2(offset(2):end);
16
17 x3 = varargin{5};
18 y3 = varargin{6};
19 x3 = x3(offset(3):end);
20 y3 = y3(offset(3):end);
21
22 % Makes U in the SVD describe the principal directions in space and
```

```
% V the principal directions in time
% 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.').';

% unmean data
X = X - means;
[u, s, v] = svd(X);

% By diagonalizing our covariance matrix, we can generate some
% completely independent component
Y = u'*X; % note we want ' not .' as here we want complex conjugate U*
principal_components = Y(1:pc, :);

% 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
% mean the data
A = A + means;
energy = sum(singularvalues(1:rankapprox))/sum(singularvalue);
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);
```

```
95
96 subplot(233)
97 plot(x3, 'r'), hold on;
98 ylim(yrange)
99 xlim([0, n])
100 title('x3')
101 plot(A(5,:), 'b')
102 legend({'original', 'low-rank approx'})
```

```
103 set(gca, 'fontsize', 15);
104
105 subplot(236)
106 plot(y3, 'r'), hold on;
107 ylim(yrange)
108 xlim([0, n])
109 title('y3')
110 plot(A(6,:), 'b')
111 legend({'original', 'low-rank approx'})
112 set(gca, 'fontsize', 15);
113
114 % plot the principal_components
115 figure(8)
116 names = [];
117 for k = 1:pc
118     plot(principal_components(k, :), 'linewidth', 2), hold on;
119     names = [names strcat("Principal Component ", num2str(k))];
120     xlabel('Time (frame)')
121     ylabel('Position')
122 end
123     xlim([0, length(principal_components)]);
124     title('Principal Components');
125     legend(names);
126     set(gca, 'fontsize', 20);
127 end
```