Joshua Whalen
AMATH 482
February 21, 2020

Assignment 3: Principle Component Analysis

Abstract

The goal of this assignment was to explore principle component analysis (PCA) given four similar yet distinctly different data sets. Each data set included video recordings of a spring attached to a paint can in harmonic vertical motion, but involved different types of interference that would affect the PCA. The four cases were as follows: ideal, noisy, horizontal translation, and horizontal translation with rotation. The method I chose for obtaining the data involved masking the window to only include the paint can, and finding the position of the max color, which should refers to the flashlight on the top of the can. Then I put the X and Y displacement vectors into an SVD function for principle component analysis. The data showed that this data collection method is not optimized for noisy conditions because it took a 4th rank (out of 6 possible ranks) approximation to capture 90% of the data. But for the ideal, horizontal, horizontal/rotational cases this data collection method was able to capture the movement in an appropriate amount of singular values.

Introduction

Principle Component Analysis (PCA) is an extremely useful tool for condensing large matrices of data into small approximations that will still capture a percentage of the variation in the data. This can be applied to systems with any level of dimensionality. By removing redundancy in the data and condensing it to a smaller the matrix, there is less computational power spent on further analysis. This method can also give insight to areas of most variance, which is important for future recording of data because more emphasis can be placed on recording these high variance areas. PCA has many applications and is often used in any data collection and processing involving high dimensional variation.

For this assignment, I tracked various movements a paint can on a spring from multiple angles: above, below, and horizontal. The harmonic movement of a weighted spring is a good example for testing PCA methods because the equation that governs the movement is simple and known (F = ma). By using PCA the movement can be empirically derived. PCA is a powerful tool because it is often applied to data sets that do not have a known governing equations, where the movement is unknown.

Theoretical Background

This section will give more information regarding the governing equation for the paint cans movement, where PCA can be extremely useful, and how PCA is calculated. For an ideal mass (paint can) on a spring moving in one direction, the equation is

$$\frac{d^2 f(t)}{dt^2} = -\omega^2 f(t)$$

The f(t) is the position of the paint can as a function of time, and ω is the angular velocity of the paint can. This ODE simplifies down to

$$f(t) = A\cos(\omega t + \omega_0)$$

The A represents the amplitude of the paint cans movement, ω_0 is the initial angular velocity, ω is the current angular velocity, and t is time. This is a very simple movement, which has an intuitive optimal camera placement for data collection. This example only needs one camera set ideally at the equilibrium point of the cans movement to see all the movement. But this example can easily become far more complex in less ideal situations where there are additional movements in other dimensions or noise from a shaking camera. This is where PCA is extremely useful because it will find the idealized movement and optimal viewing point for recording the data.

PCA is calculated using singular value decomposition (SVD). This method transforms each row in a given matrix of orthogonal data points into hyperellipses in order to break down the data and compare variance between recordings to find redundancies. A useful figure for visualizing the transform of the original matrix is shown below. Ultimately, the SVD breaks down the data by its total variance in each data set, which then can be rebuilt as a rank approximation, including less rows/columns compared to the original matrix.

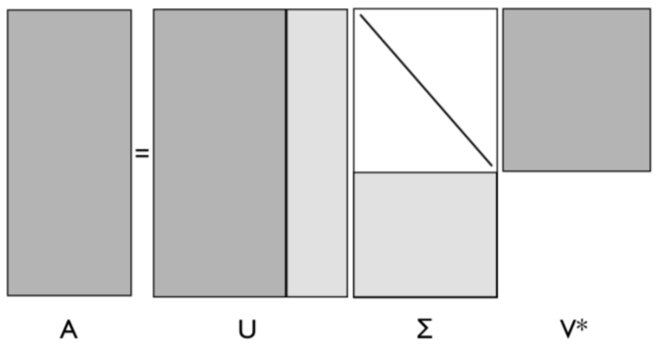
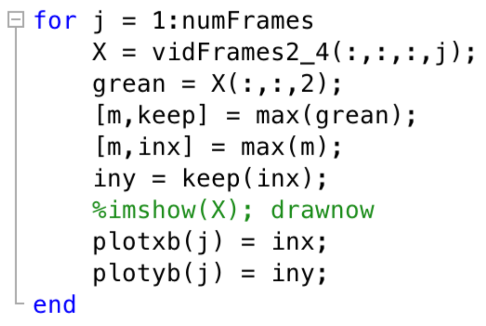


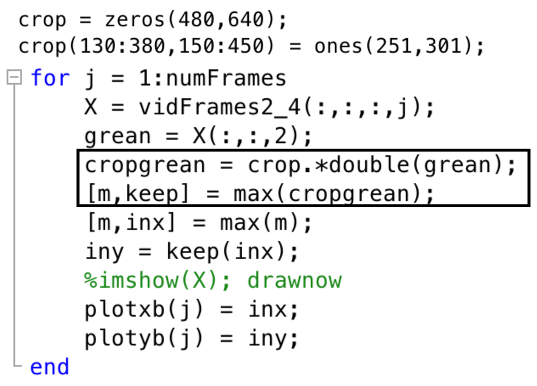
Figure 1: This figure shows a graphical representation of the relationships between outputs of the SVD (U,S,V) and the original matrix A.

Algorithm Implementation: Data Collection

This section will give an overview to the computational methods used to collect the data, the preparation of the data before SVD, and the generation of the figures shown throughout the results. The process for implementing SVD is generally the same for most data sets, but the actual obtaining of desired data can be challenging. In this case of the paint can moving on the spring, I was given 4 dimensional image data that included RGB color values that varied in both X/Y positional space and time. There was a flashlight placed on the can, which registered a max color value (255) for red, green, and blue for most points in time. This led me to try and track the overall max color value in time hopefully that it would record the position of the flashlight moving in time. However, there were many reflections of light in the recording at various locations, which also registered a 255 max color. This resulted in the recording jumping from the flashlight position, to the whiteboard, to a chair, etc. To overcome this I implemented a masking window that would change all RGB values that are not near the paint can to zero and keep the RGB values near the paint can the same. This filtered out all the reflections I did not want to track. This method worked well for the ideal case, but in the noisy case and cases with more dimensional movement the tracking wasn't as accurate. Tracking a more visible object like the white of the paint can may result in better tracking data, but this method was still functional for this example, and showed the effects of noisy and higher dimensional movement on PCA well.



```
for j = 1:numFrames
    X = vidFrames2_4(:,:,:,j);
    grean = X(:,:,2);
    [m,keep] = max(grean);
    [m,inx] = max(m);
    iny = keep(inx);
    %imshow(X); drawnow
    plotxb(j) = inx;
    plotyb(j) = iny;
end
```



```
crop = zeros(480,640);
crop(130:380,150:450) = ones(251,301);
for j = 1:numFrames
    X = vidFrames2_4(:,:,:,j);
    grean = X(:,:,2);
    cropgrean = crop.*double(grean);
    [m,keep] = max(cropgrean);
    [m,inx] = max(m);
    iny = keep(inx);
    %imshow(X); drawnow
    plotxb(j) = inx;
    plotyb(j) = iny;
end
```

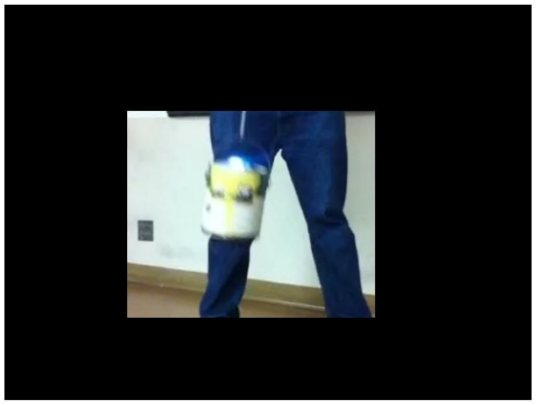


Figure 2: This figure shows the for loop used to find the max color signal and the masking window that removed the light reflections interfering with tracking of the paint can.

Algorithm Implementation: SVD Preparation and Implementation

After obtaining the data, I needed to prepare the data for SVD and plot the results from the SVD. All of the recordings for each case had a different number of frames and showed the paint can reaching its minimum position at different time steps. Therefore, I needed to sync up the recordings and make sure they had the same number of recordings. I did this by finding the minimum position of the paint can in the first 50 frames in each recording and cut out any frames before that minimum position. Then I cut ends of all the recordings to match which ever recording was shortest. This gave me 6 vectors per cases (X and Y positions for each camera angle) all with the same length and starting near the same relative position. After imputing the data into the MATLAB SVD function I plotted the position vectors in time, and the variance of the first three principle components. I also calculated the cumulative energy of each singular value (i) using the energy function below.

$$Energy = \frac{S_i^2}{\sum S^2}$$

Computational Results: Obtaining the Raw Data

This section will show the data gathered from some of the recordings of the 12 paint can videos. There are 3 angled recordings for each of the 4 cases: ideal, noisy, horizontal displacement, and horizontal displacement with rotation. Figure 3 shows the raw positional data in each case from camera angle 1. It is clear by comparing the ideal and noisy cases, that the horizontal movement from the shaky camera affected the recording greatly by adding in movement in the x direction when there was no real horizontal movement by the can. The horizontal movements in cases 3 & 4 were captured weakly, but suprisingly it seems like the rotational movement was recorded better than the horizontal displacement as seen in the defined curving strokes on either side of the main vertical displacement in case 4. Ultimately, this data collection method was satisfactory for the PCA because it captured the overall movement and subtle differences between each case. If the recording method was already perfect then there would be no point to use SVD.

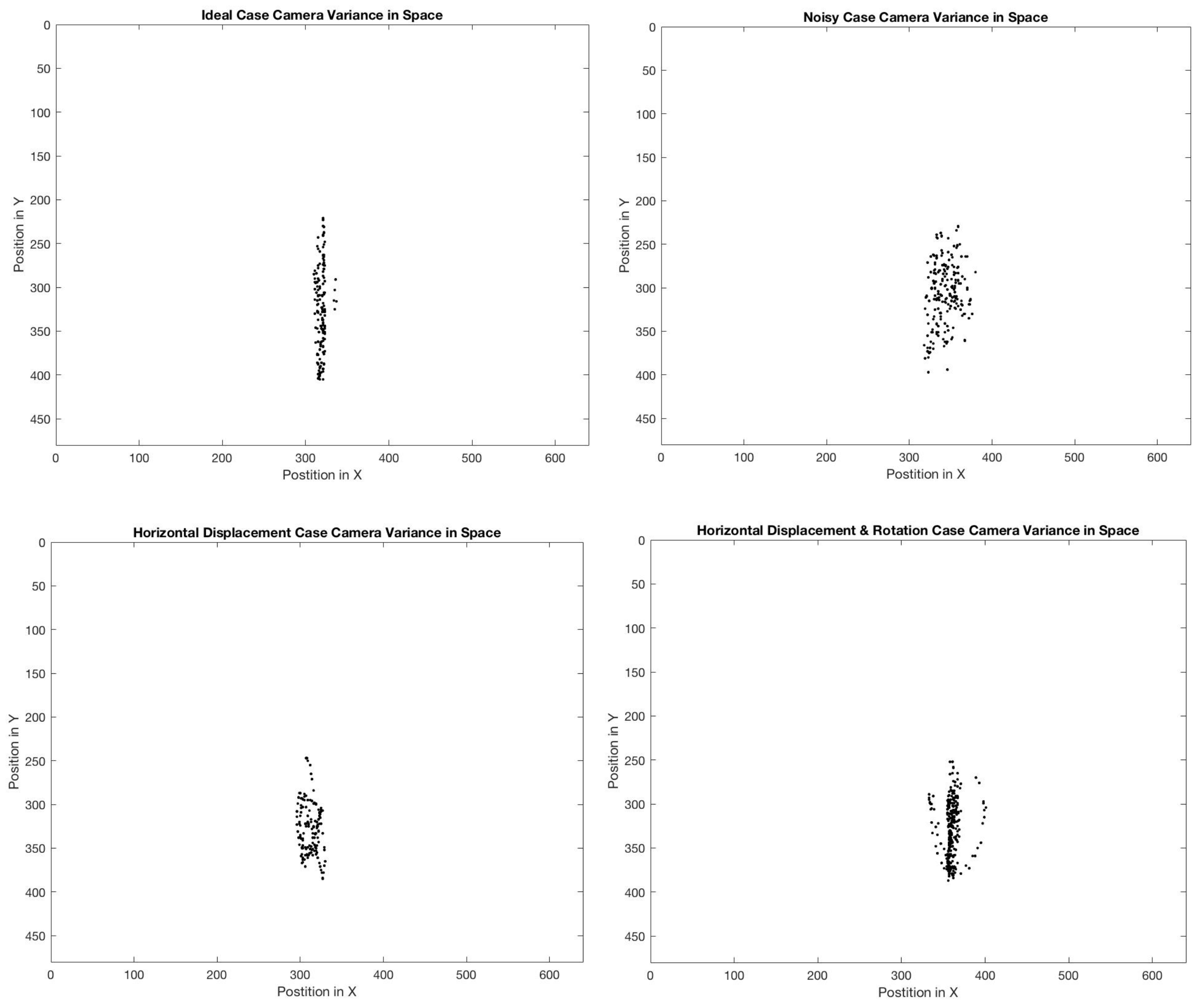


Figure 3: This figure shows the postitional data of the paint can as it was tracked in time. The ideal case (top left) has very clear movement in the Y direction with little X displacement. The noisy case (top right) has more displacement in the X direction. The horizontal displacement case with and without rotation are shown in the bottom two plots.

Computational Results: SVD Data

This section will cover the SVD data and the plots generated from the SVD. The raw positional data in each case was put into the SVD function, and then the cumulative sum of energy was plotted. In the ideal case, it shows that one singular value is enough to capture ~90% of the data. This makes sense because there is only 1 dimension of movement so most of the data beyond the first camera angle will be redundant. However, in the noisy case, it takes 4 singular values to capture the overall movement of the paint can. This shows how noise can greatly impact the ability of SVD, making it significantly less useful. It would've been best to apply a filter to reduce the affects of the shaking camera. The horizontal movement was difficult to capture with the current camera set up because there wasn't a camera angle completely perpendicular with the horizontal motion. This meant it took more input from different singular values to capture this motion. It is interesting to note that there isn't much difference when rotation was added in case 4 (only a slight decrease in between value 1 and 2). This may be because each of the camera angles were already able to capture the rotation well, so there was likely a large amount of redundant data between the recordings.

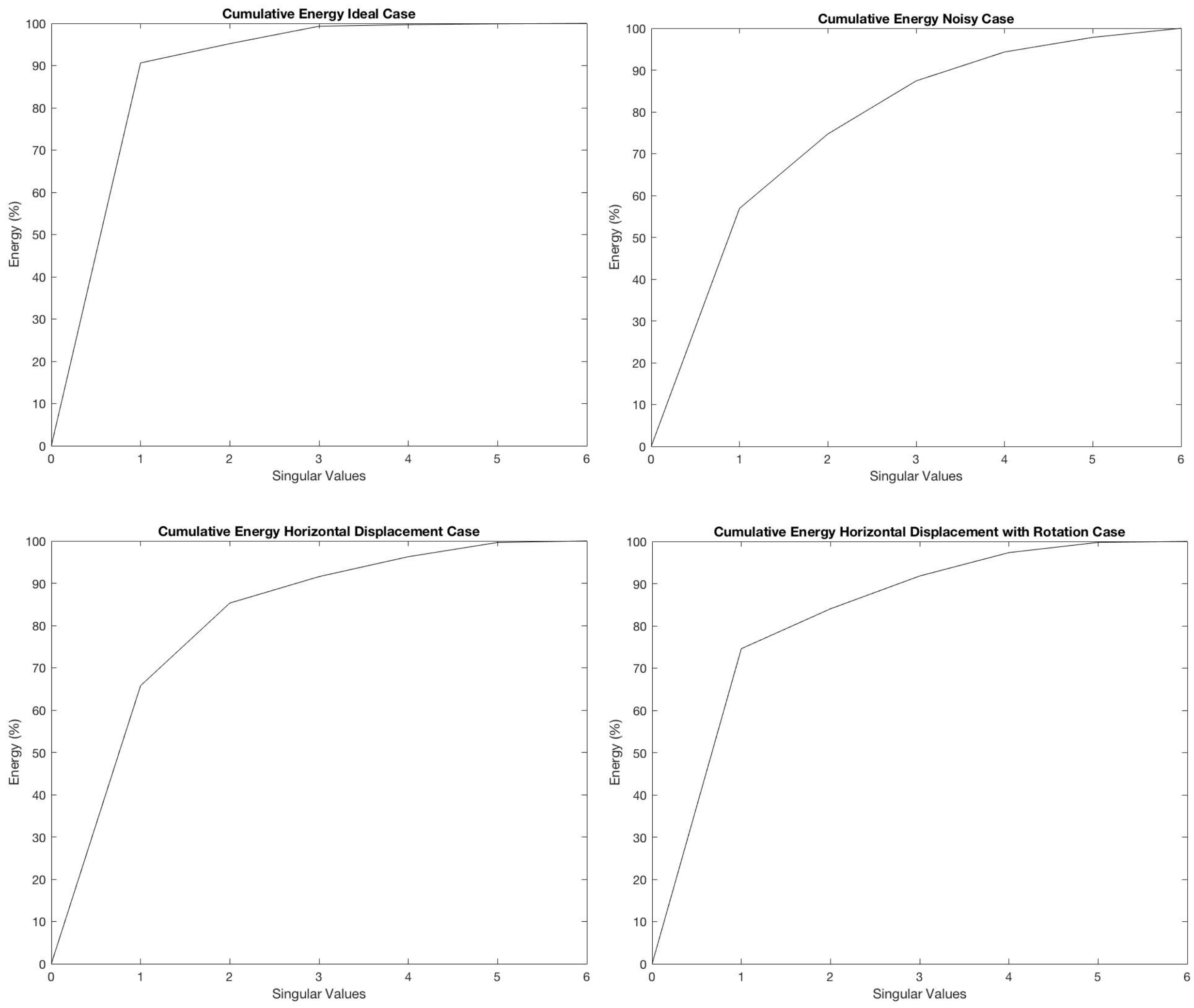


Figure 4: This graph shows the cumulative energy over the 6 singular values for each case.

In the ideal case after sync up the videos, the positional recordings over time for each camera angle are very similar as shown in figure 5. Additionally, the amplitude of variance drops dramatically after the first principle component. This explains why one singular value is able to account for 90% of the data information. As a comparison, the displacement in time and variance plots for the noisy case show much worse correlation in time, and all three principle components have a similar amplitude. This is effects the percent of energy contained in each singular value, and explains why 3 or more singular values are needed to capture the movement within the random noise.

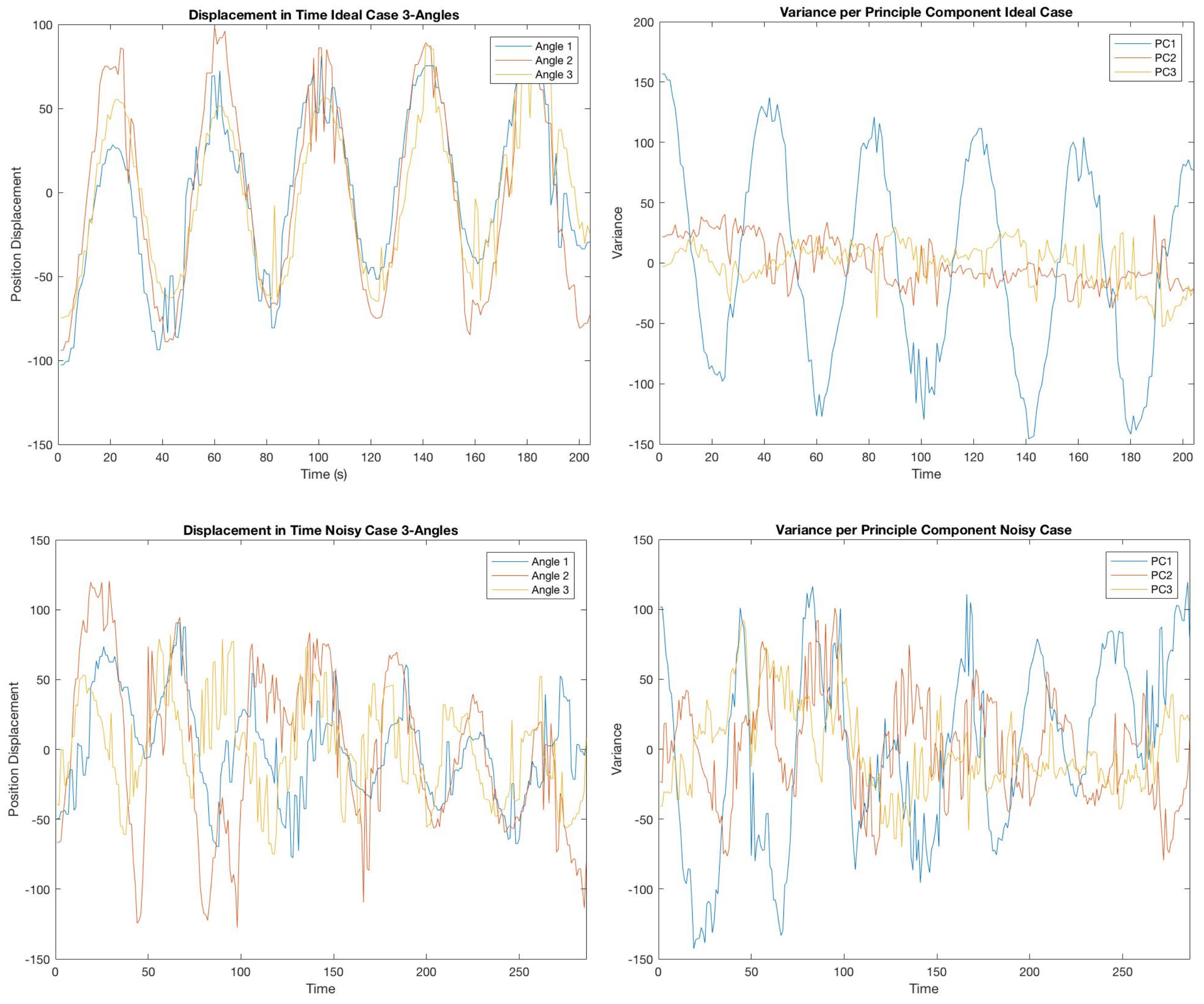


Figure 5: This figure shows the time displacement and variance per principle component for the ideal and noisy case.

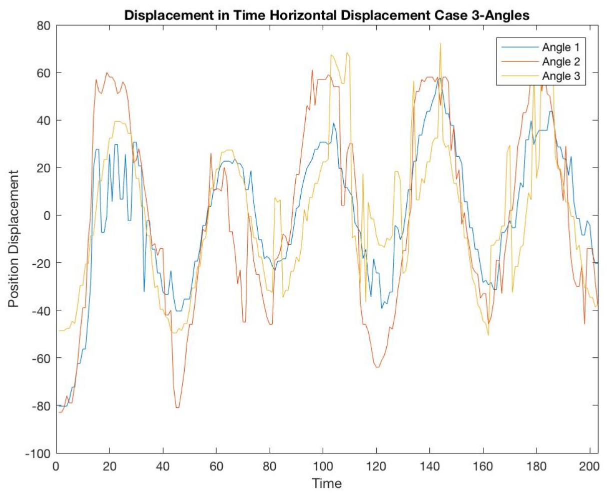
Conclusions

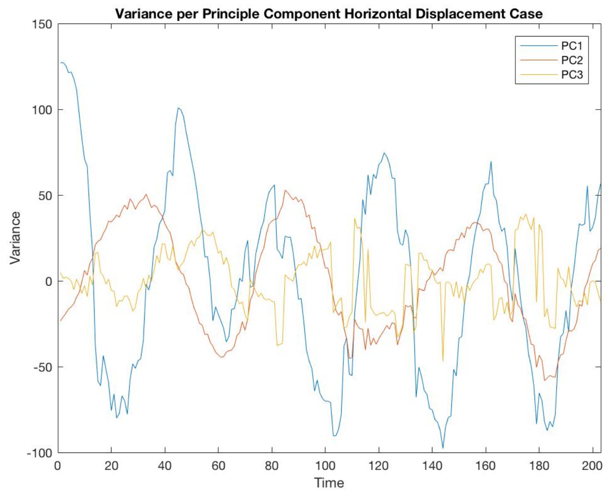
In conclusion, principle component analysis using SVD is a powerful data processing tool that allows dimensional reduction of redundant data, which is very important for analysis of larger systems that have thousands of data sets because it will greatly reduce the computational time. PCA is one of the many applications of SVD. With SVD, you can also recreate an approximation of the original data in method called proper mode decomposition (POD).

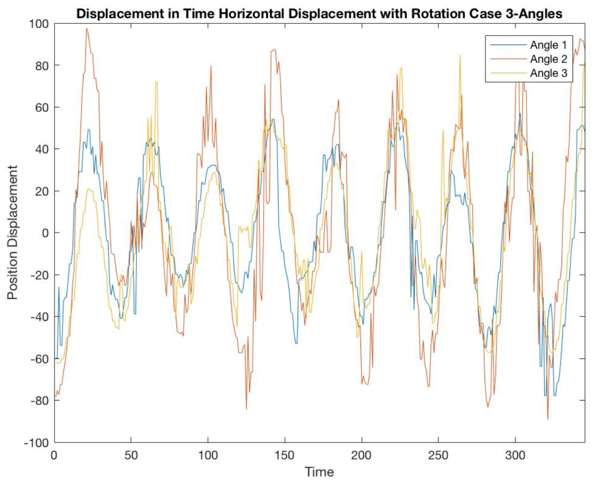
To save space, the plots for the displacement in time and variance per principle component for cases 3&4 are shown in "Appendix A"

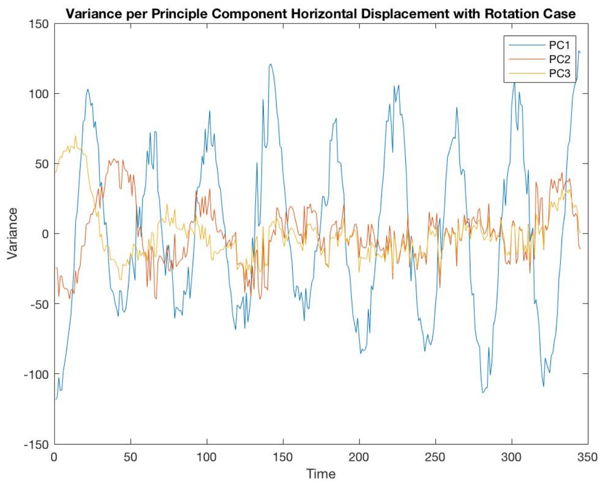
Appendix A: Extra Figures

Displacement in Time & Variance per Principle Component for Cases 3-4









Appendix B: Functions & Explanation

- Size: reports size of matrix
- SVD: calculated the single value decomposition of a given matrix of data
- Min: finds the minimum value, reports value and index
- Max: finds the maximum value, reports value and index
- Implay: plots images in time
- Imshow: show a single time point of a video

Appendix C: Code

```
%% cam 1.1
clear all; close all; clc;
load('cam1_1.mat')
%implay(vidFrames1_1)
framesize = size(vidFrames1_1);
numFrames = size(vidFrames1_1,4);

plotya = zeros(1,numFrames);
plotxa = zeros(1,numFrames);
crop = zeros(480,640);
crop(200:430,250:450) = ones(231,201);
% test = uint8(crop).*rgb2gray(vidFrames1_1(:,:,:,20));
% imshow(test)

for j = 1:numFrames
    X = vidFrames1_1(:,:,:,j);
    grean = X(:,:,2);
    cropgrean = crop.*double(grean);
    [m,keep] = max(cropgrean);
    [m,inx] = max(m);
    iny = keep(inx);
    %imshow(X); drawnow
    plotxa(j) = inx;
    plotya(j) = iny;
end
[mn, mnindx] = min(plotya(1:50));
plotya = plotya(mnindx:end);
plotxa = plotxa(mnindx:end);
figure()
plot(plotxa,plotya,'k.')
set(gca, 'YDir','reverse')
xlim([0 640])
ylim([0 480])
title('Ideal Case Camera Variance in Space')
ylabel('Position in Y')
xlabel('Postition in X')
%% cam 2.1

load('cam2_1.mat')
%implay(vidFrames1_1)
numFrames = size(vidFrames2_1,4);

plotyb = zeros(1,numFrames);
plotxb = zeros(1,numFrames);
crop = zeros(480,640);
crop(100:400,200:400) = ones(301,201);
% test = uint8(crop).*rgb2gray(vidFrames2_1(:,:,:,20));
% imshow(test)

for j = 1:numFrames
    X = vidFrames2_1(:,:,:,j);
    grean = X(:,:,2);
    cropgrean = crop.*double(grean);
    [m,keep] = max(cropgrean);
    [m,inx] = max(m);
    iny = keep(inx);
    %imshow(X); drawnow
    plotxb(j) = inx;
    plotyb(j) = iny;
end
[mn, mnindx] = min(plotyb(1:50));
plotyb = plotyb(mnindx:end);
plotxb = plotxb(mnindx:end);
figure()
plot(plotxb,plotyb,'k.')
set(gca, 'YDir','reverse')
xlim([0 640])
ylim([0 480])
%% cam 3.1

load('cam3_1.mat')
%implay(vidFrames1_1)
numFrames = size(vidFrames3_1,4);

plotyc = zeros(1,numFrames);
plotxc = zeros(1,numFrames);
crop = zeros(480,640);
crop(200:430,250:450) = ones(231,201);
% test = uint8(crop).*rgb2gray(vidFrames1_1(:,:,:,20));
% imshow(test)

for j = 1:numFrames
    X = vidFrames3_1(:,:,:,j);
    grean = X(:,:,2);
    cropgrean = crop.*double(grean);
    [m,keep] = max(cropgrean);
    [m,inx] = max(m);
    iny = keep(inx);
    %imshow(X); drawnow
    plotxc(j) = inx;
    plotyc(j) = iny;
end
[mn, mnindx] = min(plotxc(1:50));
plotyc = plotyc(mnindx:end);
plotxc = plotxc(mnindx:end);
figure()
plot(plotxc,plotyc,'k.')
set(gca, 'YDir','reverse')
xlim([0 640])
ylim([0 480])

%% SVD
svdmatrix = [plotxa(1:204)-mean(plotxa(1:204)); plotya(1:204)-
mean(plotya(1:204)); plotxb(1:204)-mean(plotxb(1:204)); plotyb(1:204)-
mean(plotyb(1:204)); plotxc(1:204)-mean(plotxc(1:204)); plotyc(1:204)-
mean(plotyc(1:204))];

[U,S,V] = svd(svdmatrix, 'econ');
Sdi = diag(S)';
energy = zeros(1,length(Sdi));
for i = 1:length(Sdi)
    energy(i) = Sdi(i)^2/(sum(Sdi.^2));
```

```
end
etot = cumsum(energy);
figure()
semilogy(etot, 'k'); drawnow
title('Cumulative Energy Ideal Case')
ylabel('Energy (%)')
xlabel('Singular Values')


figure()
plot(1:204,U(:,1:3)'*svdmatrix)
title('Variance per Principle Component Ideal Case')
ylabel('Variance')
xlabel('Time')
legend('PC1','PC2','PC3')
xlim([0 204])

figure()
plot(1:204,[svdmatrix(2,:);svdmatrix(4,:);svdmatrix(5,:)])
title('Displacement in Time Ideal Case 3-Angles')
ylabel('Position Displacement')
xlabel('Time (s)')
legend('Angle 1','Angle 2','Angle 3')
xlim([0 204])
%% cam 1.2

load('cam1_2.mat')
%implay(vidFrames1_1)
framesize = size(vidFrames1_2);
numFrames = size(vidFrames1_2,4);

plotya = zeros(1,numFrames);
plotxa = zeros(1,numFrames);
crop = zeros(480,640);
crop(200:430,250:450) = ones(231,201);
% test = uint8(crop).*vidFrames1_2(:,:,:,:);
% implay(test)

for j = 1:numFrames
    X = vidFrames1_2(:,:,:,j);
    grean = X(:,:,2);
    cropgrean = crop.*double(grean);
    [m,keep] = max(cropgrean);
    [m,inx] = max(m);
    iny = keep(inx);
    %imshow(X); drawnow
    plotxa(j) = inx;
    plotya(j) = iny;
end
[mn, mnindx] = min(plotya(1:50));
plotya = plotya(mnindx:end);
plotxa = plotxa(mnindx:end);
figure()
plot(plotxa,plotya,'k.')
set(gca, 'YDir','reverse')
xlim([0 640])
ylim([0 480])
```

```
title('Noisy Case Camera Variance in Space')
ylabel('Position in Y')
xlabel('Postition in X')
%% cam 2.2

load('cam2_2.mat')
%implay(vidFrames1_1)
framesize = size(vidFrames2_2);
numFrames = size(vidFrames2_2,4);

plotyb = zeros(1,numFrames);
plotxb = zeros(1,numFrames);
crop = zeros(480,640);
crop(80:430,150:450) = ones(351,301);
% test = uint8(crop).*vidFrames2_2(:,:,:,:);
% implay(test)

for j = 1:numFrames
    X = vidFrames2_2(:,:,:,j);
    grean = X(:,:,2);
    cropgrean = crop.*double(grean);
    [m,keep] = max(cropgrean);
    [m,inx] = max(m);
    iny = keep(inx);
    %imshow(X); drawnow
    plotxb(j) = inx;
    plotyb(j) = iny;
end
[mn, mnindx] = min(plotyb(1:50));
plotyb = plotyb(mnindx:end);
plotxb = plotxb(mnindx:end);
figure()
plot(plotxb,plotyb,'k.')
set(gca, 'YDir','reverse')
xlim([0 640])
ylim([0 480])

%% cam 3.2

load('cam3_2.mat')
%implay(vidFrames1_1)
framesize = size(vidFrames3_2);
numFrames = size(vidFrames3_2,4);

plotyc = zeros(1,numFrames);
plotxc = zeros(1,numFrames);
crop = zeros(480,640);
crop(200:430,250:450) = ones(231,201);
% test = uint8(crop).*vidFrames3_2(:,:,:,:);
% implay(test)

for j = 1:numFrames
    X = vidFrames3_2(:,:,:,j);
    grean = X(:,:,2);
    cropgrean = crop.*double(grean);
    [m,keep] = max(cropgrean);
    [m,inx] = max(m);
    iny = keep(inx);
    %imshow(X); drawnow
    plotxc(j) = inx;
    plotyc(j) = iny;
end
[mn, mnindx] = min(plotyc(1:50));
plotyc = plotyc(mnindx:end);
plotxc = plotxc(mnindx:end);
figure()
plot(plotxc,plotyc,'k.')
set(gca, 'YDir','reverse')
xlim([0 640])
ylim([0 480])

%% SVD
svdmatrix = [plotxa(1:286)-mean(plotxa(1:286)); plotya(1:286)-
mean(plotya(1:286)); plotxb(1:286)-mean(plotxb(1:286)); plotyb(1:286)-
mean(plotyb(1:286)); plotxc(1:286)-mean(plotxc(1:286)); plotyc(1:286)-
mean(plotyc(1:286))];

[U,S,V] = svd(svdmatrix, 'econ');
Sdi = diag(S)';
energy = zeros(1,length(Sdi));
for i = 1:length(Sdi)
    energy(i) = Sdi(i)^2/(sum(Sdi.^2));
end
etot = cumsum(energy);
figure()
semilogy(etot, 'k'); drawnow
title('Cumulative Energy Noisy Case')
ylabel('Energy (%)')
xlabel('Singular Values')


figure()
plot(1:286,U(:,1:3)'*svdmatrix)
title('Variance per Principle Component Noisy Case')
ylabel('Variance')
xlabel('Time')
legend('PC1','PC2','PC3')
xlim([0 286])

figure()
plot(1:286,[svdmatrix(2,:);svdmatrix(4,:);svdmatrix(5,:)])
title('Displacement in Time Noisy Case 3-Angles')
ylabel('Position Displacement')
xlabel('Time')
legend('Angle 1','Angle 2','Angle 3')
xlim([0 286])
%% cam 1.3

load('cam1_3.mat')
%implay(vidFrames1_1)
framesize = size(vidFrames1_3);
numFrames = size(vidFrames1_3,4);

plotya = zeros(1,numFrames);
```

```
plotxa = zeros(1,numFrames);
crop = zeros(480,640);
crop(200:430,250:450) = ones(231,201);
% test = uint8(crop).*(vidFrames1_3(:,:,:,:));
% implay(test)

for j = 1:numFrames
    X = vidFrames1_3(:,:,:,j);
    grean = X(:,:,2);
    cropgrean = crop.*double(grean);
    [m,keep] = max(cropgrean);
    [m,inx] = max(m);
    iny = keep(inx);
    %imshow(X); drawnow
    plotxa(j) = inx;
    plotya(j) = iny;
end
[mn, mnindx] = min(plotya(1:50));
plotya = plotya(mnindx:end);
plotxa = plotxa(mnindx:end);
figure()
plot(plotxa,plotya,'k.')
set(gca, 'YDir','reverse')
xlim([0 640])
ylim([0 480])
title('Horizontal Displacement Case Camera Variance in Space')
ylabel('Position in Y')
xlabel('Postition in X')
%% cam 2.3

load('cam2_3.mat')
%implay(vidFrames1_1)
framesize = size(vidFrames2_3);
numFrames = size(vidFrames2_3,4);

plotyb = zeros(1,numFrames);
plotxb = zeros(1,numFrames);
crop = zeros(480,640);
crop(170:400,150:450) = ones(231,301);
% test = uint8(crop).*(vidFrames2_3(:,:,:,:));
% implay(test)

for j = 1:numFrames
    X = vidFrames2_3(:,:,:,j);
    grean = X(:,:,2);
    cropgrean = crop.*double(grean);
    [m,keep] = max(cropgrean);
    [m,inx] = max(m);
    iny = keep(inx);
    %imshow(X); drawnow
    plotxb(j) = inx;
    plotyb(j) = iny;
end
[mn, mnindx] = min(plotyb(1:50));
plotyb = plotyb(mnindx:end);
plotxb = plotxb(mnindx:end);
figure()
```

```
plot(plotxb,plotyb,'k.')
set(gca, 'YDir','reverse')
xlim([0 640])
ylim([0 480])
%% cam 3.3

load('cam3_3.mat')
%implay(vidFrames1_1)
framesize = size(vidFrames3_3);
numFrames = size(vidFrames3_3,4);

plotyc = zeros(1,numFrames);
plotxc = zeros(1,numFrames);
crop = zeros(480,640);
crop(200:430,250:450) = ones(231,201);
% test = uint8(crop).*(vidFrames3_3(:,:,:,:));
% implay(test)

for j = 1:numFrames
    X = vidFrames3_3(:,:,:,j);
    grean = X(:,:,2);
    cropgrean = crop.*double(grean);
    [m,keep] = max(cropgrean);
    [m,inx] = max(m);
    iny = keep(inx);
    %imshow(X); drawnow
    plotxc(j) = inx;
    plotyc(j) = iny;
end
[mn, mnindx] = min(plotxc(1:50));
plotyc = plotyc(mnindx:end);
plotxc = plotxc(mnindx:end);
figure()
plot(plotxc,plotyc,'k.')
set(gca, 'YDir','reverse')
xlim([0 640])
ylim([0 480])
%% SVD
svdmatrix = [plotxa(1:203)-mean(plotxa(1:203)); plotya(1:203)-
mean(plotya(1:203)); plotxb(1:203)-mean(plotxb(1:203)); plotyb(1:203)-
mean(plotyb(1:203)); plotxc(1:203)-mean(plotxc(1:203)); plotyc(1:203)-
mean(plotyc(1:203))];

[U,S,V] = svd(svdmatrix, 'econ');
Sdi = diag(S)';
energy = zeros(1,length(Sdi));
for i = 1:length(Sdi)
    energy(i) = Sdi(i)^2/(sum(Sdi.^2));
end
etot = cumsum(energy);
figure()
semilogy(etot, 'k'); drawnow
title('Cumulative Energy Horizontal Displacement Case')
ylabel('Energy (%)')
xlabel('Singular Values')

figure()
```

```
plot(1:203,U(:,1:3)'*svdmatrix)
title('Variance per Principle Component Horizontal Displacement Case')
ylabel('Variance')
xlabel('Time')
legend('PC1','PC2','PC3')
xlim([0 203])

figure()
plot(1:203,[svdmatrix(2,:);svdmatrix(4,:);svdmatrix(5,:)])
title('Displacement in Time Horizontal Displacement Case 3-Angles')
ylabel('Position Displacement')
xlabel('Time')
legend('Angle 1','Angle 2','Angle 3')
xlim([0 203])
%% cam 1.4

load('cam1_4.mat')
%implay(vidFrames1_1)
framesize = size(vidFrames1_4);
numFrames = size(vidFrames1_4,4);

plotya = zeros(1,numFrames);
plotxa = zeros(1,numFrames);
crop = zeros(480,640);
crop(200:430,250:450) = ones(231,201);
% test =uint8(crop).*(vidFrames1_4(:,:,:,:));
% implay(test)

for j = 1:numFrames
    X = vidFrames1_4(:,:,:,j);
    grean = X(:,:,2);
    cropgrean = crop.*double(grean);
    [m,keep] = max(cropgrean);
    [m,inx] = max(m);
    iny = keep(inx);
    %imshow(X); drawnow
    plotxa(j) = inx;
    plotya(j) = iny;
end
[mn, mnindx] = min(plotya(1:50));
plotya = plotya(mnindx:end);
plotxa = plotxa(mnindx:end);
figure()
plot(plotxa,plotya,'k.')
set(gca, 'YDir','reverse')
xlim([0 640])
ylim([0 480])
title('Horizontal Displacement & Rotation Case Camera Variance in Space')
ylabel('Position in Y')
xlabel('Postition in X')
%% cam 2.4

load('cam2_4.mat')
%implay(vidFrames1_1)
framesize = size(vidFrames2_4);
numFrames = size(vidFrames2_4,4);
```

```
plotyb = zeros(1,numFrames);
plotxb = zeros(1,numFrames);
crop = zeros(480,640);
crop(130:380,150:450) = ones(251,301);
% test = uint8(crop).*(vidFrames2_4(:,:,:,:));
% implay(test)

for j = 1:numFrames
    X = vidFrames2_4(:,:,:,j);
    grean = X(:,:,2);
    cropgrean = crop.*double(grean);
    [m,keep] = max(cropgrean);
    [m,inx] = max(m);
    iny = keep(inx);
    %imshow(X); drawnow
    plotxb(j) = inx;
    plotyb(j) = iny;
end
[mn, mnindx] = min(plotyb(1:50));
plotyb = plotyb(mnindx:end);
plotxb = plotxb(mnindx:end);
figure()
plot(plotxb,plotyb,'k.')
set(gca, 'YDir','reverse')
xlim([0 640])
ylim([0 480])
%% cam 3.4

load('cam3_4.mat')
%implay(vidFrames1_1)
framesize = size(vidFrames3_4);
numFrames = size(vidFrames3_4,4);

plotyc = zeros(1,numFrames);
plotxc = zeros(1,numFrames);
crop = zeros(480,640);
crop(100:330,300:500) = ones(231,201);
% test = uint8(crop).*(vidFrames3_4(:,:,:,:));
% implay(test)

for j = 1:numFrames
    X = vidFrames3_4(:,:,:,j);
    grean = X(:,:,2);
    cropgrean = crop.*double(grean);
    [m,keep] = max(cropgrean);
    [m,inx] = max(m);
    iny = keep(inx);
    %imshow(X); drawnow
    plotxc(j) = inx;
    plotyc(j) = iny;
end
[mn, mnindx] = min(plotxc(1:100));
plotyc = plotyc(mnindx:end);
plotxc = plotxc(mnindx:end);
figure()
plot(plotxc,plotyc,'k.')
set(gca, 'YDir','reverse')
```

```
xlim([0 640])
ylim([0 480])

%% SVD
svdmatrix = [plotxa(1:345)-mean(plotxa(1:345)); plotya(1:345)-
mean(plotya(1:345)); plotxb(1:345)-mean(plotxb(1:345)); plotyb(1:345)-
mean(plotyb(1:345)); plotxc(1:345)-mean(plotxc(1:345)); plotyc(1:345)-
mean(plotyc(1:345))];

[U,S,V] = svd(svdmatrix, 'econ');
Sdi = diag(S)';
energy = zeros(1,length(Sdi));
for i = 1:length(Sdi)
    energy(i) = Sdi(i)^2/(sum(Sdi.^2));
end
etot = cumsum(energy);
figure()
semilogy(etot, 'k'); drawnow
title('Cumulative Energy Horizontal Displacement with Rotation Case')
ylabel('Energy (%)')
xlabel('Singular Values')
figure()
plot(1:345,U(:,1:3)'*svdmatrix)
title('Variance per Principle Component Horizontal Displacement with Rotation
Case')
ylabel('Variance')
xlabel('Time')
legend('PC1','PC2','PC3')
%%
figure()
plot(1:345,[svdmatrix(2,:);svdmatrix(4,:);svdmatrix(5,:)])
title('Displacement in Time Horizontal Displacement with Rotation Case 3-
Angles')
ylabel('Position Displacement')
xlabel('Time')
legend('Angle 1','Angle 2','Angle 3')
xlim([0 345])
```