AMATH 582 Homework 3: Principal Component Analysis (PCA)

Sara L. Daley

2/21/2020

Abstract

The goal of this paper is to effectively use Principal Component Analysis in an experiment containing video data for numerous cases that is oversampled to understand the signals underneath.

1 Introduction and Overview

What happens when you're given data that appears complex and/or random, but yet do not know the governing equations behind the data? What if your data was also redundant? Can we identify the signals with maximum variance? With Principal Component Analysis (PCA), we can take dimensional reductions of the dynamics and behavior in the data to analyze the systems underneath. This homework focuses on real video footage filming a mass-spring system. Before we dive into what the PCA means, we first describe the experiment and cleaning the data.

1.1 Experiment

Figure 1 shows a basic implementation of the experiment. We had three cameras recording video via smartphones, of the mass-spring system. In reality the mass was a paint can, with a spring and someone holding the top of the spring system. On top of the paint can, there was a flashlight and we were asked to use the data from each of the cameras to track the paint can and use PCA for analysis. We had four tests or cases to consider.

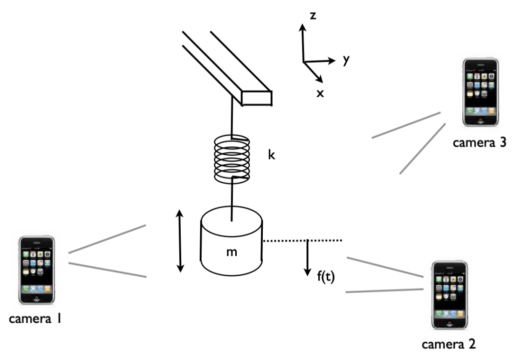


Figure 1: Visual of Experiment Setup[1]

The first case was the ideal- where those holding the phones kept as still as possible and we see the paint can moving up and down with very little to no swaying in the x-y direction (we see only movement in the z direction)[1]. The second case added noise to the videos. In essence, those holding the cameras were shaking at various intensities and times. The third case added horizontal displacement to the paint can. Meaning the paint can moved not only in the z direction but also in the x-y direction. The fourth case added rotation to the horizontal displacement. Meaning that the paint can moved as in the last case but it was also spinning. We notice that because there are three different cameras recording the same object, the data between the cameras will be dependent, and this redundancy essentially defines what it means to have an oversampling of data. In order to specify how redundant these datasets are, consider the covariance between them. The higher the covariance, the higher the redundancy. We can find redundancy with,

$$\sigma^2_{\mathbf{ab}} = \frac{1}{n-1}\mathbf{ab}^T \tag{1}$$

where **a** and **b** are row vectors of our dataset, and n is the length of the set (or number of frames).

[1]We will note now that the third camera was recording at a different orientation than the others.

1.2 Cleaning and Centering the Data

Before we can use PCA to analyze the data, we have to clean and normalize it. To filter we first turned the image into black and white, which removes saturation and hue. This was done because we wanted to make use of the flashlight on the top of the can. The second filtering step was to crop the video to focus on only the can as much as possible. For the ideal case we could crop quite a bit, but for the noise and horizontal displacements we had to keep more of the image space. The third filtering step focused on pixel intensity. Because we wanted to make use of the flashlight, we focused on all pixels above a certain intensity threshold. For example, in the first case, first camera we focused on pixels ≥ 252. It is important to note that each camera is different, and they recorded at different rates. Thus, with every camera and every experiment this number was different. After performing this layer of filtering, we normalized. Because we have to use all three cameras together to analyze each test case, we want them to start at similar intervals. In order to do this, we looped through the first 20 frames, isolated the minimum and started each position vector at the minimum. To center we subtracted the vector by its mean. At this point every camera still has varying frame lengths, so before running PCA analysis we make them the same length as the minimum for that particular test case.

2 Theoretical Background

The main point to refer to is basis transformations. If we start with a vector $\mathbf{x}$ and scale it by $\mathbf{A}$ the result is a new magnitude and direction. A singular value decomposition (SVD) is a factorization of a matrix into a number of constitutive components all of which have a specific meaning in applications[1]. The full SVD decomposition takes the form[1]

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

where

$$\mathbf{U} \in \mathbb{C}^{\mathbf{mxm}} \text{ is unitary} \tag{3}$$

$$\mathbf{V} \in \mathbb{C}^{\mathbf{nxn}} \text{ is unitary} \tag{4}$$

$$\mathbf{\Sigma} \in \mathbb{R}^{\mathbf{mxn}} \text{ is diagonal} \tag{5}$$

This is saying that we can decompose a matrix that we don't know anything about ($\mathbf{A}$ here) and find out information from its displacement ($\mathbf{U}$), singular values ($\mathbf{\Sigma}$)), and direction ($\mathbf{V}$). The theorem behind the SVD:

Theorem 1. *Every matrix* $\mathbf{A} \in \mathbb{C}^{\mathbf{mxn}}$ *has a singular value decomposition 2. Furthermore, the singular values* σ_j *are uniquely determined, and, if* $\mathbf{A}$ *is square and the* σ_j *distinct, the singular vectors* $\mathbf{u_j}$ *and* $\mathbf{v_j}$ *are uniquely determined up to complex signs (complex scalar factors of absolute value 1).[1]*

Note that if `rank(A)` is r, then there are r nonzero singular values ($\mathbf{\Sigma}$). According to the book on Page 387, "the SVD gives a type of least-square fitting algorithm, allowing us to project the matrix onto low dimensional representations in a formal, algorithmic way."[1]

Principal Component Analysis is a variant of Singular Value Decomposition. We want to look at the principal components; therefore we want to focus on the variance of the data.

3 Algorithm Implementation and Development

Please refer to Algorithm 1

4 Computational Results

We completed PCA analysis on each of the four cases. Each case contained three videos that were cleaned (minimally) and standardized. These results were then compiled together to create the $\mathbf{X}$ matrix, and used that to calculate the robust SVD using the **inexact_alm_rpca** algorithm[3]. To explain more clearly, our $\mathbf{X}$ vector is a matrix that contains our $\mathbf{x}$ and $\mathbf{y}$ data from each camera. For each Case,

$$\mathbf{X} = [\mathbf{x}_a; \mathbf{y}_a; \mathbf{x}_b; \mathbf{y}_b; \mathbf{x}_c; \mathbf{y}_c] where \tag{6}$$

$$Camera1 = (\mathbf{x}_a, \mathbf{y}_a), Camera2 = (\mathbf{x}_b, \mathbf{y}_b), Camera3 = (\mathbf{x}_c, \mathbf{y}_c) \tag{7}$$

Recall in Equation 2 that the SVD decomposes our signal into singular values ($\mathbf{\Sigma}$)), displacement ($\mathbf{U}$), and direction ($\mathbf{V}$). Thus, to document the results we plotted all components together. Consider Figure 2, the first

Algorithm 1: PCA on Three Video Files

```
Import cam<CamNum_CaseNum>.mat
Set the video up in MATLAB so it can be watched, using the loop:
for $j = 1 : numFrames$ do
  mv(j).cdata = vidFrames(:,:,:,j)
  mov(k).colormap = []
end for
for $j = 1 : numFrames$ do
  Frame to Image Space, frame2im
  Image to black & white, rgb2gray
  Zoom in on the can, Image(ypixels,xpixels)
  Find all pixels above threshold save x & y vectors,
  find(ZoomImage>=thresh)
  Intermediate step: imshow to watch movie
  xcamNum = savedx(j)
  ycamNum = savedy(j)
end for
Line up each data set to start at a minimum value using two steps:
1. Use ind2sub to find min of ycamNum(1:20)
2. Recast x to be: xcamNum = xcamNum(ymin:end) and repeat for y
Center the signals around 0. This will make comparisons easier to see:
x = x-mean(x), repeat for y
Repeat for other cameras in Cases
find minimum size of each of the recording lengths
create $\mathbf{X}$ = [xcam1; ycam1; xcam2; ycam2; ...]
set lambda
calculate robust $\mathbf{X}$ using inexact_alm_rpca algorithm
calculate $\mathbf{U}$, $\Sigma$, and $\mathbf{V}$
calculate Energy for every mode
create necessary plots
```

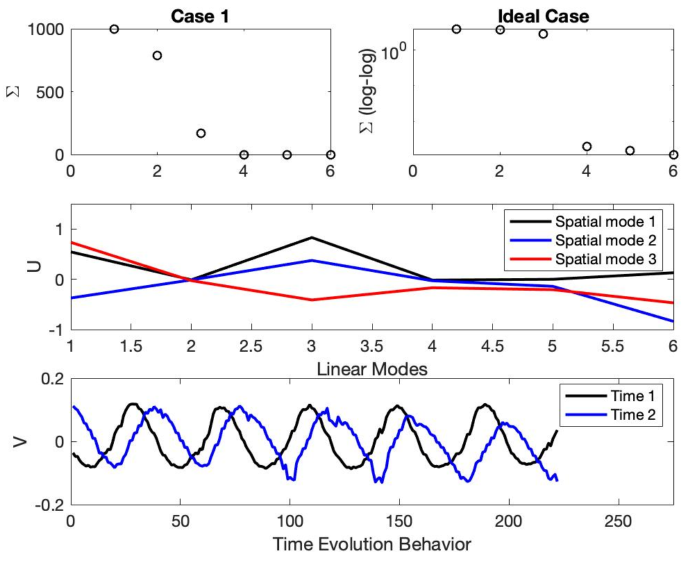


Figure 2: PCA Analysis of Case 1: Ideal

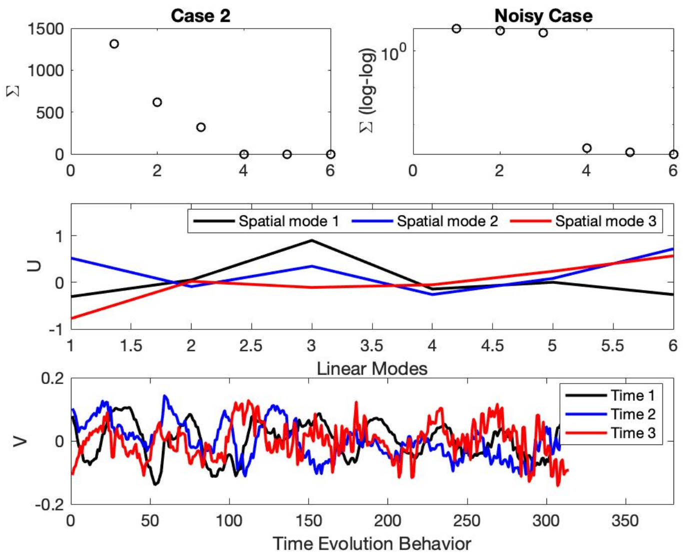


Figure 3: PCA Analysis of Case 2: Noise

two plots are the singular values (the absolute values of Equation 2, **A**'s

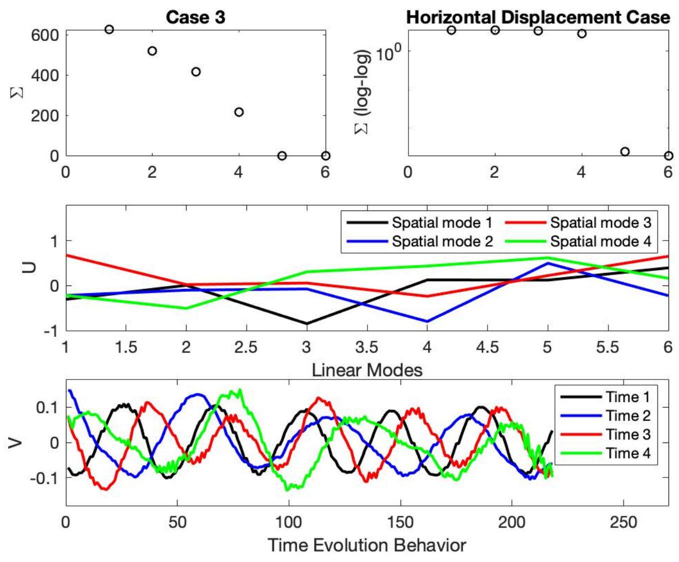


Figure 4: PCA Analysis of Case 3: Horizontal Displacement

eigenvalues); we see that there are three modes that are captured with the third mode being minimal, and the fourth through sixth being zero. The middle plot is the displacement along each of the modes. The last plot shows the direction over time. We removed the third time component as it was noisy and made the plot harder to read. This is due to the fact that the third mode was much smaller than the first two and therefore less significant. We see that the displacement of the paint can follows a wave, which is what we expect. We expect to see the paint can follow

$$-\omega^2(A\cos(\omega t+\omega_0)). \tag{8}$$

For Case 2, Figure 3, we notice three main modes again, and the time evolution behavior of the displacement is much harder to see. You cannot tell that the can is moving along Equation 8. This implies that the data needs to be cleaned more. All we have done is crop and pull out pixels greater than a given threshold. A smoothing filter should also be applied. We attempted applying a Gaussian filter to the data, but could not narrow in on a correct form for it. The filter either pulled everything out- meaning we got a black screen video with nothing to see, or let everything in- meaning our data was the exact same as before. Due to time constraints, we could not follow this

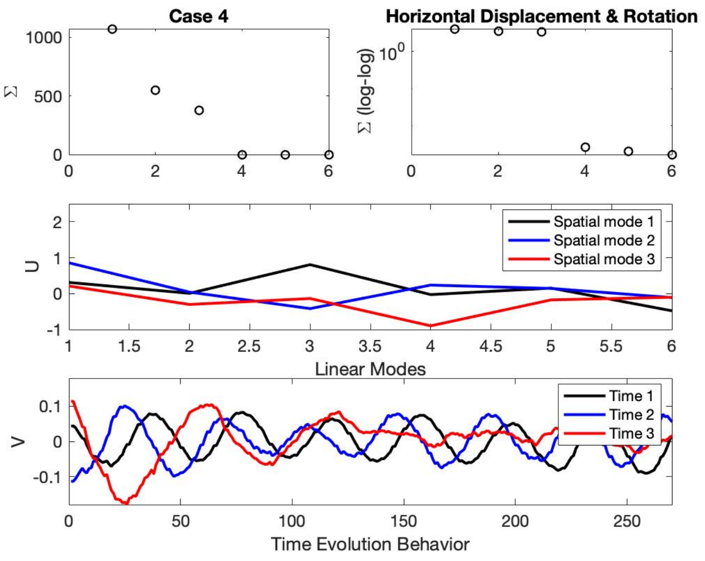


Figure 5: PCA Analysis of Case 4: Horizontal Displacement & Rotation

further, but think that would be helpful for this case.

For Case 3, Figure 4, horizontal displacement was added. Thus, the can is moving up and down, and side to side. We would expect the displacement of all modes to follow a wave, and we do.

For Case 4, Figure 5, it follows Case 3; however, the can is also spinning. This is inherently harder to track a moving object. Instead of focusing on following a single point on the object, we had to widen the amount of pixels let through to try to track the *entire* object. This also let in a small amount of noise in the background. The results of this test are interesting though, in that it is only three modes, where Case 3 had four. The first two modes track the paint can very well.

At this point, we now want to switch gears and talk about the difference between the Robust PCA algorithm general PCA. The algorithm "separates the data into low-rank and sparse components"[1]. The best implementation will have a high percentage in fewer modes, and the robust method extracts meaningful, low-rank matrix from the full matrix, **X**. If you look at Figure 6, with the Robust method, we see a dramatic improvement in containing the variance (or another way of putting it would be the percentage of energy produced). In every case we see that 100% of variance is contained before

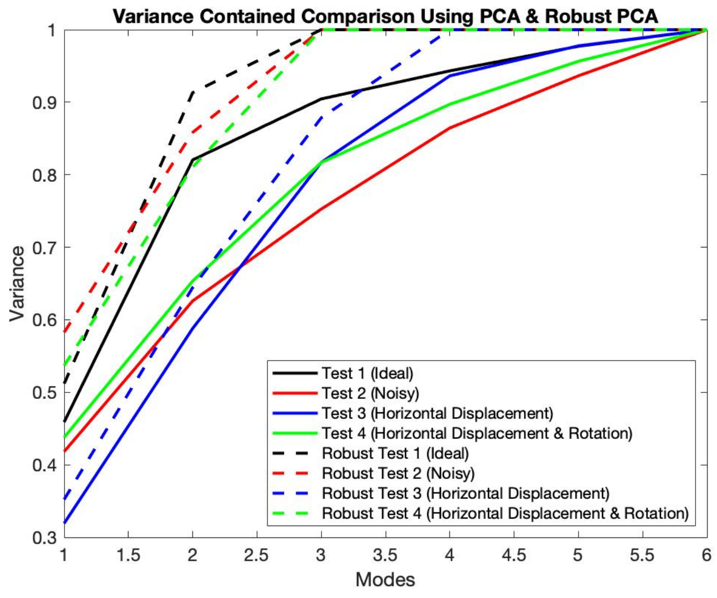


Figure 6: Percentage of Energy each of the Modes produced, comparing PCA (solid lines) and Robust PCA (dashed lines).

the last mode (mode 6). That was not the case for PCA. We even notice that in the first mode, all cases scored higher. In Case 2, the Noisy Test, we see 18% improvement in the first mode alone. What should be mentioned is that Case 2 (noise) is no longer has the worst variance, Case 3 (horizontal displacement) does. Even though it didn't appear to be a good filter when looking at displacement in Figure 3, it did decompose the data well.

5 Summary and Conclusions

Principal Component Analysis has worked remarkably well in tracking the moving object for many different cases. It has correctly found the cosine (Equation 8) wave for three of the four cases, along with the maximum variance. We also looked at how implementing the Robust PCA dramatically improved results by acting as an advanced filter, which changed our opinion on how PCA handled the Noisy Case.

References

[1] Jose Nathan Kutz. *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data*, Oxford University Press, 2013.

[2] `https://www.mathworks.com/help/matlab/index.html`

[3] `https://github.com/andrewssobral/lrslibrary` `https://www.mathworks.com/matlabcentral/fileexchange/48404-lrslibrary`

Appendices

Please refer to my Github repo for all code and related documents.

A MATLAB Functions

- `[a,b] = find(zoom1_1>=252)`

  this returns the values (not indicies)
- `frame2im`: returns image data of movie frame [2]
- `rgb2grey`: eliminates hue and saturation- keeping illumination [2]
- `repmat(A,M,N)`: tiles an array (through replication) by creating matrix B of M x N tiling of A[2].
- `svd(X)`: produces diag matrix S, along with unitary U and V.
- `inexact_alm_rpca`: here we used it to create our 'idealized' $\mathbf{X}$, inputs are $\mathbf{X}$ and λ.

B MATLAB Codes

Included here is my code for this homework set, though I do call the `inexact_alm_rpca` function (which calls two other supporting scripts). These codes will be found on my Github repo, referenced above.

Table of Contents

```
close all; clc; clear

% This code pulls in 12 camN_N.mat (where N=1,2,3) files found here:
%https://drive.google.com/drive/
folders/1SQ77P5t5RUWCSucmk4jPFbufFMX8VrJG
% Due to size, it is not put on GitHub.
```

TEST 1: Ideal Case, Camera 1

```
load cam1_1.mat
% Reads in as (height, width, color pane, frame number)
numFrames1_1 = size(vidFrames1_1,4);
for k = 1:numFrames1_1
    mov(k).cdata = vidFrames1_1(:,:,:,k);
    mov(k).colormap = [];
end

for j=1:numFrames1_1
    X1_1=frame2im(mov(j));
    X1_1 = rgb2gray(X1_1); % convert to grayscale
    zoom1_1 = X1_1(180:430,220:390); % zoom around can
    %imshow(zoom1_1); drawnow
    [a,b] = find(zoom1_1>=252);
    x1_1(j) = mean(a);
    y1_1(j) = mean(b);
end

[M11 I11] = min(y1_1(1:20));
[ymin1_1] = ind2sub(size(y1_1),I11); % Find first min to line up
```

```
x1_1 = x1_1(ymin1_1:end);
y1_1 = y1_1(ymin1_1:end); % Recast vectors to start at min

x1_1 = x1_1 - mean(x1_1);
y1_1 = y1_1 - mean(y1_1); % Subtract mean to align around 0
```

TEST 1: Ideal Case, Camera 2

```
load cam2_1.mat
numFrames2_1 = size(vidFrames2_1,4);
for k = 1:numFrames2_1
    mov(k).cdata = vidFrames2_1(:,:,:,k);
    mov(k).colormap = [];
end

for j=1:numFrames2_1
    X2_1=frame2im(mov(j));
    X2_1 = rgb2gray(X2_1); % convert to grayscale
    zoom2_1 = X2_1(80:390,200:400);
    %imshow(zoom2_1); drawnow %; h=gca; h.Visible='On'
    [a,b] = find(zoom2_1>=250); % keep bright pixels
    x2_1(j) = mean(a);
    y2_1(j) = mean(b);
end

[M21 I21] = min(y2_1(1:20));
[ymin2_1] = ind2sub(size(y2_1),I21);

x2_1 = x2_1(ymin2_1:end);
y2_1 = y2_1(ymin2_1:end);

x2_1 = x2_1 - mean(x2_1);
y2_1 = y2_1 - mean(y2_1);
```

TEST 1: Ideal Case, Camera 3

```
load cam3_1.mat
numFrames3_1 = size(vidFrames3_1,4);
for k = 1:numFrames3_1
    mov(k).cdata = vidFrames3_1(:,:,:,k);
    mov(k).colormap = [];
end

for j=1:numFrames3_1
    X3_1=frame2im(mov(j));
    X3_1 = rgb2gray(X3_1); % convert to grayscale
    zoom3_1 = X3_1(240:340,250:500);
    %imshow(zoom3_1); drawnow %; h=gca; h.Visible='On'
    [a,b] = find(zoom3_1>=248); % keep bright pixels
    x3_1(j) = mean(a);
    y3_1(j) = mean(b);
```

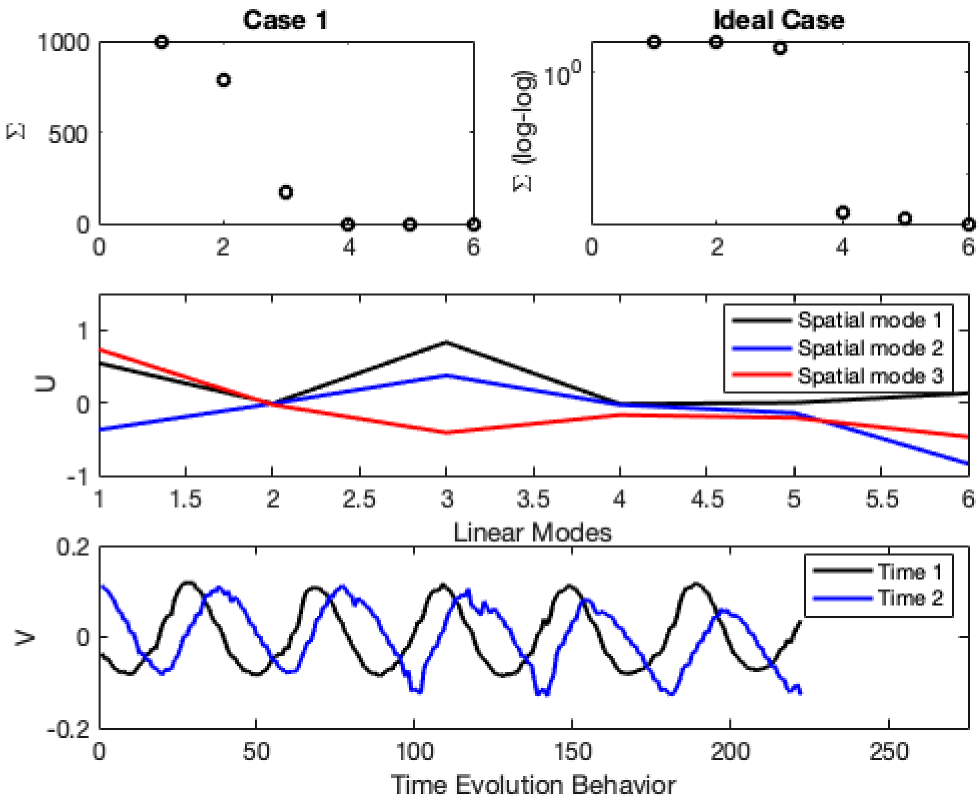
```
end

[M31 I31] = min(y3_1(1:20));
[ymin3_1] = ind2sub(size(y3_1),I31);

x3_1 = x3_1(ymin3_1:end);
y3_1 = y3_1(ymin3_1:end);

x3_1 = x3_1 - mean(x3_1);
y3_1 = y3_1 - mean(y3_1);
```

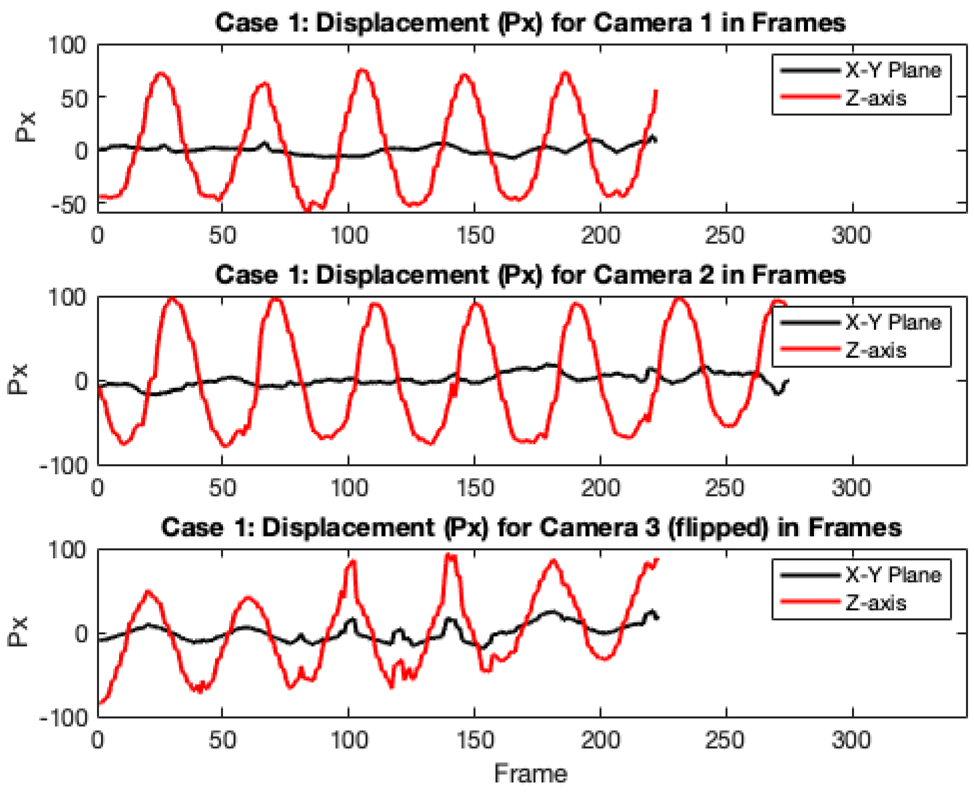
TEST 1: Standardize, PCA, & Plot

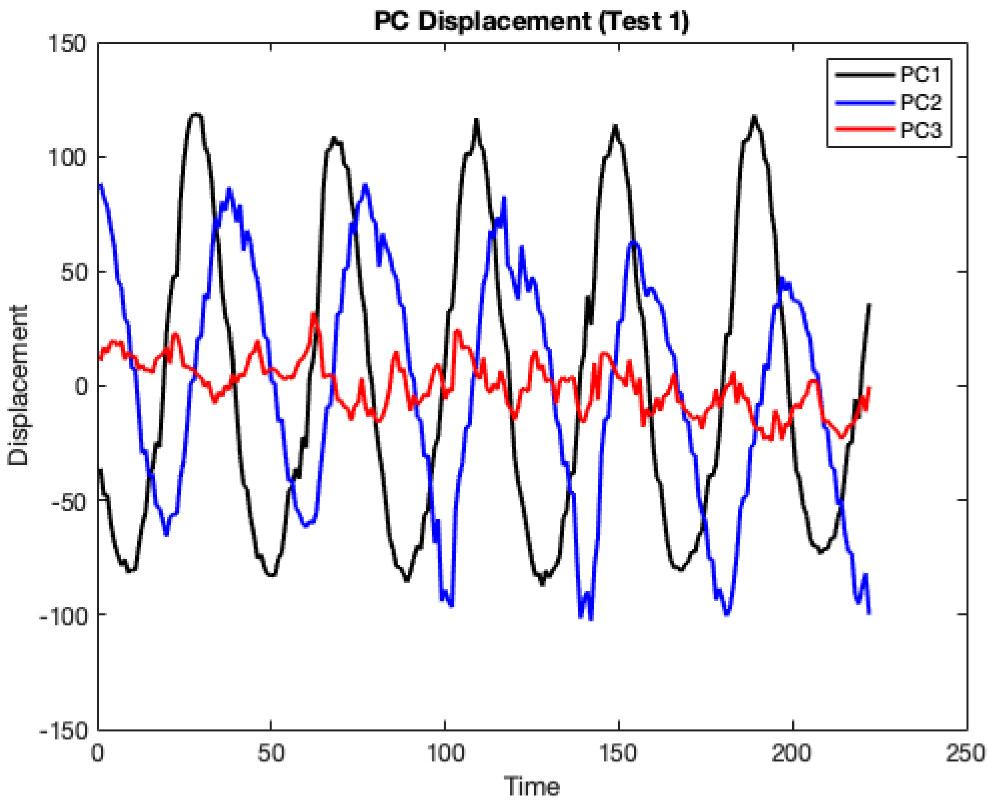
```
if ~isequal(size(x1_1), size(y1_1)) % Make sure each camera x,y size
 match
    disp('Go back to camera1_1')
end
if ~isequal(size(x2_1), size(y2_1))
    disp('Go back to camera2_1')
end
if ~isequal(size(x3_1), size(y3_1))
    disp('Go back to camera3_1')
end

size1_1 = size(y1_1,2); size2_1 = size(y2_1,2); size3_1 =
 size(y3_1,2);
sizer = min([size1_1 size2_1 size3_1]); % take the min length
ind = 1:sizer;
ax = max([size1_1 size2_1 size3_1]);

uu = linspace(1,6,6);
vv = linspace(1,length(ind),length(ind));

X1 = [x1_1(ind); y1_1(ind); x2_1(ind); y2_1(ind); x3_1(ind);
 y3_1(ind)];
[m,n] = size(X1);
mn = mean(X1,2);
X1 = X1 - repmat(mn,1,n);
[u,s,v] = svd(X1);
sigma1 = diag(s);
Y1 = u'*X1;

energy1_1=sigma1(1)/sum(sigma1);
energy2_1=sum(sigma1(1:2))/sum(sigma1);
energy3_1=sum(sigma1(1:3))/sum(sigma1);
energy4_1=sum(sigma1(1:4))/sum(sigma1);
energy5_1=sum(sigma1(1:5))/sum(sigma1);
energy6_1=sum(sigma1(1:6))/sum(sigma1);

% Robust PCA
lambda = 0.2;
[X1R,X12R] = inexact_alm_rpca(X1.',lambda);
[ur, sr, vr] = svd(X1R.');
```

```
sigma1R = diag(sr);
Y1R = ur'*X1R.';
%X1R = X1R'; Y1R = Y1R';

energy1_1R=sigma1R(1)/sum(sigma1R);
energy2_1R=sum(sigma1R(1:2))/sum(sigma1R);
energy3_1R=sum(sigma1R(1:3))/sum(sigma1R);
energy4_1R=sum(sigma1R(1:4))/sum(sigma1R);
energy5_1R=sum(sigma1R(1:5))/sum(sigma1R);
energy6_1R=sum(sigma1R(1:6))/sum(sigma1R);

figure(1)
subplot(3,2,1), plot(sigma1R, 'ko', 'Linewidth', [1.2])
set(gca,'Fontsize',[12])
%text(3.3,800,{'Singular Values', '(Energy)'}, 'Fontsize',[12])
ylabel('\Sigma'); title('Case 1')

subplot(3,2,2), semilogy(sigma1R,'ko','Linewidth',[1.2])
set(gca,'Fontsize',[12])
%text(3.3,10^(2),'log-log (Energy)','Fontsize',[12])
ylabel('\Sigma (log-log)'); title('Ideal Case')

subplot(3,1,2)
plot(uu,ur(:,1),'k',uu,ur(:,2),'b',uu,ur(:,3),'r','Linewidth',[2])
set(gca,'Fontsize',[12]); axis([1 6 -1 1.5])
legend('Spatial mode 1','Spatial mode 2', 'Spatial mode 3', ...
    'Location','northeast')
xlabel('Linear Modes'); ylabel('U')

subplot(3,1,3)
plot(vv,vr(:,1),'k',vv,vr(:,2),'b','Linewidth',[2])
set(gca,'Fontsize',[12]); axis([0 275 -0.2 0.2])
legend('Time 1','Time 2', 'Location','northeast')
xlabel('Time Evolution Behavior'); ylabel('V')

figure(2)
subplot(3,1,1)
plot(vv,y1_1,'k',vv,x1_1,'r','Linewidth',[2])
set(gca,'Fontsize',[12]); xlim([0 (ax+70)])
title('Case 1: Displacement (Px) for Camera 1 in Frames')
ylabel('Px'); legend('X-Y Plane','Z-axis')
subplot(3,1,2)
plot(1:length(y2_1),y2_1,'k',1:length(y2_1),x2_1,'r','Linewidth',[2])
set(gca,'Fontsize',[12]); xlim([0 (ax+70)])
title('Case 1: Displacement (Px) for Camera 2 in Frames')
ylabel('Px'); legend('X-Y Plane','Z-axis')
subplot(3,1,3)
plot(1:length(y3_1),x3_1,'k',1:length(y3_1),y3_1,'r','Linewidth',[2])
set(gca,'Fontsize',[12]); xlim([0 (ax+70)])
title('Case 1: Displacement (Px) for Camera 3 (flipped) in Frames')
xlabel('Frame'); ylabel('Px'); legend('X-Y Plane','Z-axis')

figure(3)
plot(vv,Y1R(1,:),'k',vv,Y1R(2,:),'b',vv,Y1R(3,:),'r','Linewidth',[2])
```

```
set(gca,'Fontsize',[12]);legend('PC1', 'PC2', 'PC3')
xlabel('Time'); ylabel('Displacement'); title('PC Displacement (Test
 1)')

#svd 10 r(A) 3 |E|_0 480 stopCriterion 0.0098807
#svd 20 r(A) 3 |E|_0 670 stopCriterion 2.8986e-05
#svd 30 r(A) 3 |E|_0 677 stopCriterion 8.3829e-08
```




Case 1: Displacement (Px) for Camera 1 in Frames
Case 1: Displacement (Px) for Camera 2 in Frames
Case 1: Displacement (Px) for Camera 3 (flipped) in Frames
Px
Frame
X-Y Plane
Z-axis


PC Displacement (Test 1)
Displacement
Time
PC1
PC2
PC3

TEST 2: Noisy Case, Camera 1

```
load cam1_2.mat

numFrames1_2 = size(vidFrames1_2,4);
% kx = size(vidFrames1_2,2); ky = size(vidFrames1_2,1);
% [Kx, Ky] = meshgrid(kx, ky);
% w = 10^(-4);


for k = 1:numFrames1_2
    mov(k).cdata = vidFrames1_2(:,:,:,k);
    mov(k).colormap = [];
end

for j=1:numFrames1_2
    X1_2=frame2im(mov(j));
    X1_2 = rgb2gray(X1_2); % convert to grayscale
    zoom1_2 = X1_2(180:430,220:430); % zoom around can
%     if (j==1)
%         filt = exp(-w*((Kx-159.29).^2 + (Ky-100.61).^2));
%     else
%         filt = exp(-w*((Kx-x1_2(j-1)).^2 + (Ky-y1_2(j-1)).^2));
%     end
%     zoom1_2filt = zoom1_2 * uint8(filt);
    %imshow(zoom1_2); drawnow
    [a,b] = find(zoom1_2>=252);
    %imshow(zoom1_2>=252); drawnow
    x1_2(j) = mean(a);
    y1_2(j) = mean(b);

end

[M12 I12] = min(y1_2(1:20));
[ymin1_2] = ind2sub(size(y1_2),I12); % Find first min to line up

x1_2 = x1_2(ymin1_2:end);
y1_2 = y1_2(ymin1_2:end); % Recast vectors to start at min

x1_2 = x1_2 - mean(x1_2);
y1_2 = y1_2 - mean(y1_2); % Subtract mean to align around 0
```

TEST 2: Noisy Case, Camera 2

```
load cam2_2.mat

numFrames2_2 = size(vidFrames2_2,4);
for k = 1:numFrames2_2
    mov(k).cdata = vidFrames2_2(:,:,:,k);
    mov(k).colormap = [];
end
```

```
for j=1:numFrames2_2
    X2_2=frame2im(mov(j));
    X2_2 = rgb2gray(X2_2); % convert to grayscale
    zoom2_2 = X2_2(60:420,200:420);
    %imshow(zoom2_2>=250); drawnow %; h=gca; h.Visible='On'
    [a,b] = find(zoom2_2>=250); % keep bright pixels
    x2_2(j) = mean(a);
    y2_2(j) = mean(b);
end

[M22 I22] = min(y2_2(1:20));
[ymin2_2] = ind2sub(size(y2_2),I22);

x2_2 = x2_2(ymin2_2:end);
y2_2 = y2_2(ymin2_2:end);

x2_2 = x2_2 - mean(x2_2);
y2_2 = y2_2 - mean(y2_2);
```

TEST 2: Noisy Case, Camera 3

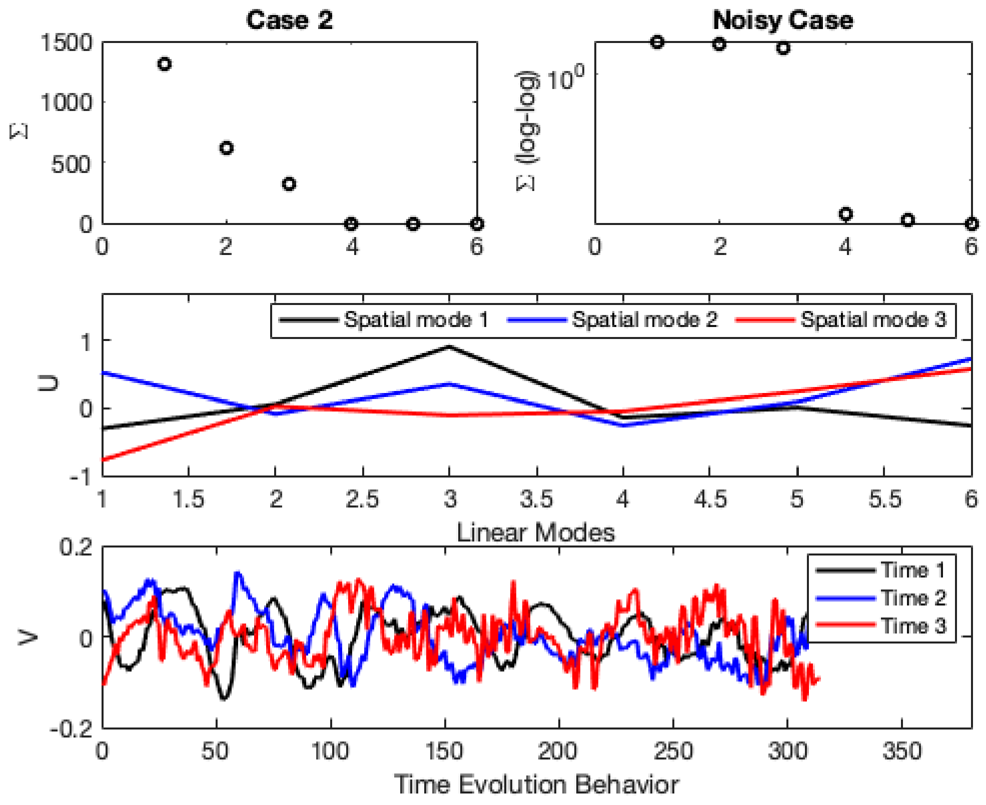
```
load cam3_2.mat
numFrames3_2 = size(vidFrames3_2,4);
for k = 1:numFrames3_2
    mov(k).cdata = vidFrames3_2(:,:,:,k);
    mov(k).colormap = [];
end

for j=1:numFrames3_2
    X3_2=frame2im(mov(j));
    X3_2 = rgb2gray(X3_2); % convert to grayscale
    zoom3_2 = X3_2(200:360,250:500);
    %imshow(zoom3_2>=246); drawnow %; h=gca; h.Visible='On'
    [a,b] = find(zoom3_2>=246); % keep bright pixels
    x3_2(j) = mean(a);
    y3_2(j) = mean(b);
end

[M32 I32] = min(y3_2(1:20));
[ymin3_2] = ind2sub(size(y3_2),I32);

x3_2 = x3_2(ymin3_2:end);
y3_2 = y3_2(ymin3_2:end);

x3_2 = x3_2 - mean(x3_2);
y3_2 = y3_2 - mean(y3_2);
```

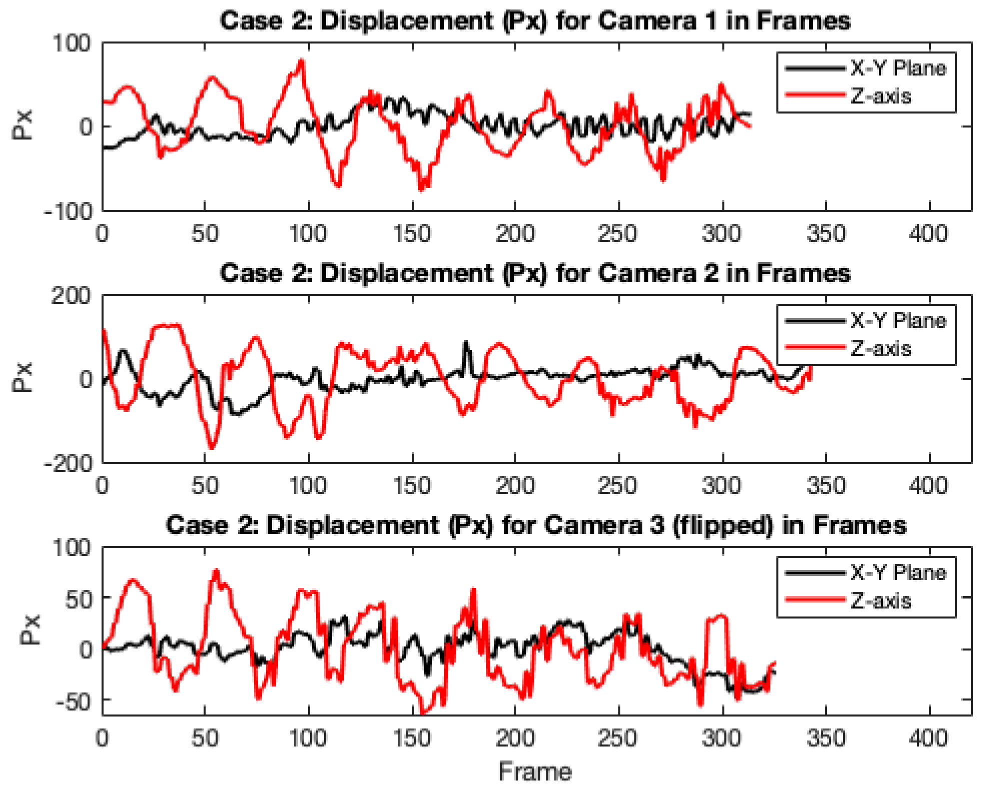
TEST 2: Standardize, PCA, & Plot

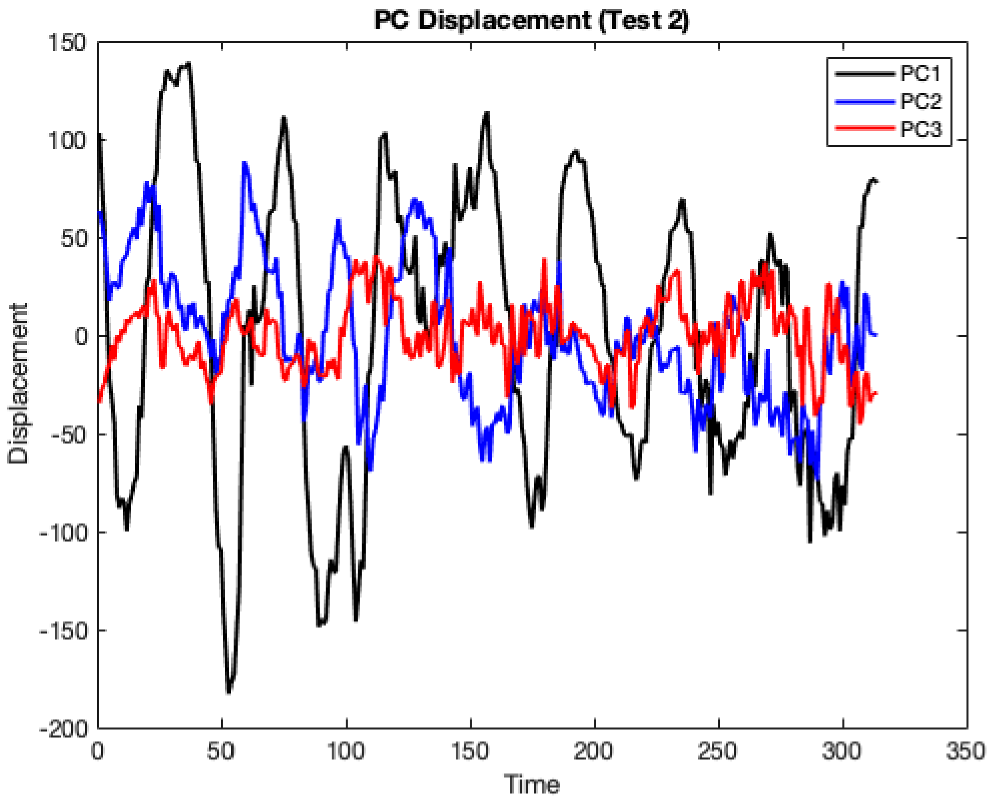
```
if ~isequal(size(x1_2), size(y1_2)) % Make sure each camera x,y size
 match
    disp('Go back to camera1_1')
```

```
end
if ~isequal(size(x2_2), size(y2_2))
    disp('Go back to camera2_1')
end
if ~isequal(size(x3_2), size(y3_2))
    disp('Go back to camera3_1')
end

size1_2 = size(y1_2,2); size2_2 = size(y2_2,2); size3_2 =
 size(y3_2,2);
sizer = min([size1_2 size2_2 size3_2]); % take the min length
ind = 1:sizer;
ax = max([size1_2 size2_2 size3_2]);

uu = linspace(1,6,6);
vv = linspace(1,length(ind),length(ind));

X2 = [x1_2(ind); y1_2(ind); x2_2(ind); y2_2(ind); x3_2(ind);
 y3_2(ind)];
[m,n] = size(X2);
mn = mean(X2,2);
X2 = X2 - repmat(mn,1,n);
[u,s,v] = svd(X2);
sigma2 = diag(s);
Y2 = u'*X2;

energy1_2=sigma2(1)/sum(sigma2);
energy2_2=sum(sigma2(1:2))/sum(sigma2);
energy3_2=sum(sigma2(1:3))/sum(sigma2);
energy4_2=sum(sigma2(1:4))/sum(sigma2);
energy5_2=sum(sigma2(1:5))/sum(sigma2);
energy6_2=sum(sigma2(1:6))/sum(sigma2);

% Robust PCA
lambda = 0.2;
[X2R,X22R] = inexact_alm_rpca(X2.',lambda);
[ur, sr, vr] = svd(X2R.');
sigma2R = diag(sr);
Y2R = ur'*X2R.';
%X1R = X1R'; Y1R = Y1R';

energy1_2R=sigma2R(1)/sum(sigma2R);
energy2_2R=sum(sigma2R(1:2))/sum(sigma2R);
energy3_2R=sum(sigma2R(1:3))/sum(sigma2R);
energy4_2R=sum(sigma2R(1:4))/sum(sigma2R);
energy5_2R=sum(sigma2R(1:5))/sum(sigma2R);
energy6_2R=sum(sigma2R(1:6))/sum(sigma2R);

figure(4)
subplot(3,2,1), plot(sigma2R, 'ko', 'Linewidth', [1.2])
set(gca,'Fontsize',[12]) % energy of each diag variance
%text(3.3,1200,{'Singular Values', '(Energy)'}, 'Fontsize',[12])
ylabel('\Sigma'); title('Case 2');
```

```
subplot(3,2,2), semilogy(sigma2R,'ko','Linewidth',[1.2])
set(gca,'Fontsize',[12])
%text(3.3,10^(2),'log-log (Energy)','Fontsize',[12])
ylabel('\Sigma (log-log)'); title('Noisy Case');

subplot(3,1,2)
plot(uu,ur(:,1),'k',uu,ur(:,2),'b',uu,ur(:,3),'r','Linewidth',[2])
set(gca,'Fontsize',[12]); axis([1 6 -1 1.7])
legend('Spatial mode 1','Spatial mode 2', 'Spatial mode 3', ...
    'Location','northeast','Orientation','horizontal')
xlabel('Linear Modes'); ylabel('U')

subplot(3,1,3)
plot(vv,vr(:,1),'k',vv,vr(:,2),'b',vv,vr(:,3),'r','Linewidth',[2])
set(gca,'Fontsize',[12]); axis([0 380 -0.2 0.2])
legend('Time 1','Time 2', 'Time 3','Location','northeast')
xlabel('Time Evolution Behavior'); ylabel('V')

figure(5)
subplot(3,1,1)
plot(vv,y1_2,'k',vv,x1_2,'r','Linewidth',[2])
set(gca,'Fontsize',[12]); xlim([0 (ax+70)])
title('Case 2: Displacement (Px) for Camera 1 in Frames')
ylabel('Px'); legend('X-Y Plane','Z-axis')
subplot(3,1,2)
plot(1:length(y2_2),y2_2,'k',1:length(y2_2),x2_2,'r','Linewidth',[2])
set(gca,'Fontsize',[12]); xlim([0 (ax+70)])
title('Case 2: Displacement (Px) for Camera 2 in Frames')
ylabel('Px'); legend('X-Y Plane','Z-axis')
subplot(3,1,3)
plot(1:length(y3_2),x3_2,'k',1:length(y3_2),y3_2,'r','Linewidth',[2])
set(gca,'Fontsize',[12]); xlim([0 (ax+70)])
title('Case 2: Displacement (Px) for Camera 3 (flipped) in Frames')
xlabel('Frame'); ylabel('Px'); legend('X-Y Plane','Z-axis')

figure(6)
plot(vv,Y2R(1,:),'k',vv,Y2R(2,:),'b',vv,Y2R(3,:),'r','Linewidth',[2])
set(gca,'Fontsize',[12]),legend('PC1', 'PC2','PC3')
xlabel('Time'); ylabel('Displacement'); title('PC Displacement (Test
 2)')

#svd 10 r(A) 3 |E|_0 885 stopCriterion 0.01075
#svd 20 r(A) 3 |E|_0 958 stopCriterion 1.5622e-05
#svd 24 r(A) 3 |E|_0 958 stopCriterion 4.4997e-08
```


Case 2
Noisy Case
Σ
Σ (log-log)
10^0
Spatial mode 1
Spatial mode 2
Spatial mode 3
U
Linear Modes
Time 1
Time 2
Time 3
V
Time Evolution Behavior


Case 2: Displacement (Px) for Camera 1 in Frames
Case 2: Displacement (Px) for Camera 2 in Frames
Case 2: Displacement (Px) for Camera 3 (flipped) in Frames
X-Y Plane
Z-axis
Px
Frame



TEST 3: Horizontal Displacement, Camera 1

```
load cam1_3.mat

numFrames1_3 = size(vidFrames1_3,4);

for k = 1:numFrames1_3
    mov(k).cdata = vidFrames1_3(:,:,:,k);
    mov(k).colormap = [];
end

for j=1:numFrames1_3
    X1_3=frame2im(mov(j));
    X1_3 = rgb2gray(X1_3); % convert to grayscale
    zoom1_3 = X1_3(180:430,220:430); % zoom around can
    %imshow(zoom1_3>=252); drawnow
    [a,b] = find(zoom1_3>=252);
    %imshow(zoom1_3>=252); drawnow
    x1_3(j) = mean(a);
    y1_3(j) = mean(b);

end

[M13 I13] = min(y1_3(1:20));
[ymin1_3] = ind2sub(size(y1_3),I13); % Find first min to line up
```

```
x1_3 = x1_3(ymin1_3:end);
y1_3 = y1_3(ymin1_3:end); % Recast vectors to start at min

x1_3 = x1_3 - mean(x1_3);
y1_3 = y1_3 - mean(y1_3); % Subtract mean to align around 0
```

TEST 3: Horizontal Displacement, Camera 2

```
load cam2_3.mat

numFrames2_3 = size(vidFrames2_3,4);
for k = 1:numFrames2_3
    mov(k).cdata = vidFrames2_3(:,:,:,k);
    mov(k).colormap = [];
end

for j=1:numFrames2_3
    X2_3=frame2im(mov(j));
    X2_3 = rgb2gray(X2_3); % convert to grayscale
    zoom2_3 = X2_3(140:420,200:420);
    %imshow(zoom2_3>=250); drawnow %; h=gca; h.Visible='On'
    [a,b] = find(zoom2_3>=250); % keep bright pixels
    x2_3(j) = mean(a);
    y2_3(j) = mean(b);
end

[M23 I23] = min(y2_3(1:20));
[ymin2_3] = ind2sub(size(y2_3),I23);

x2_3 = x2_3(ymin2_3:end);
y2_3 = y2_3(ymin2_3:end);

x2_3 = x2_3 - mean(x2_3);
y2_3 = y2_3 - mean(y2_3);
```

TEST 3: Horizontal Displacement, Camera 3

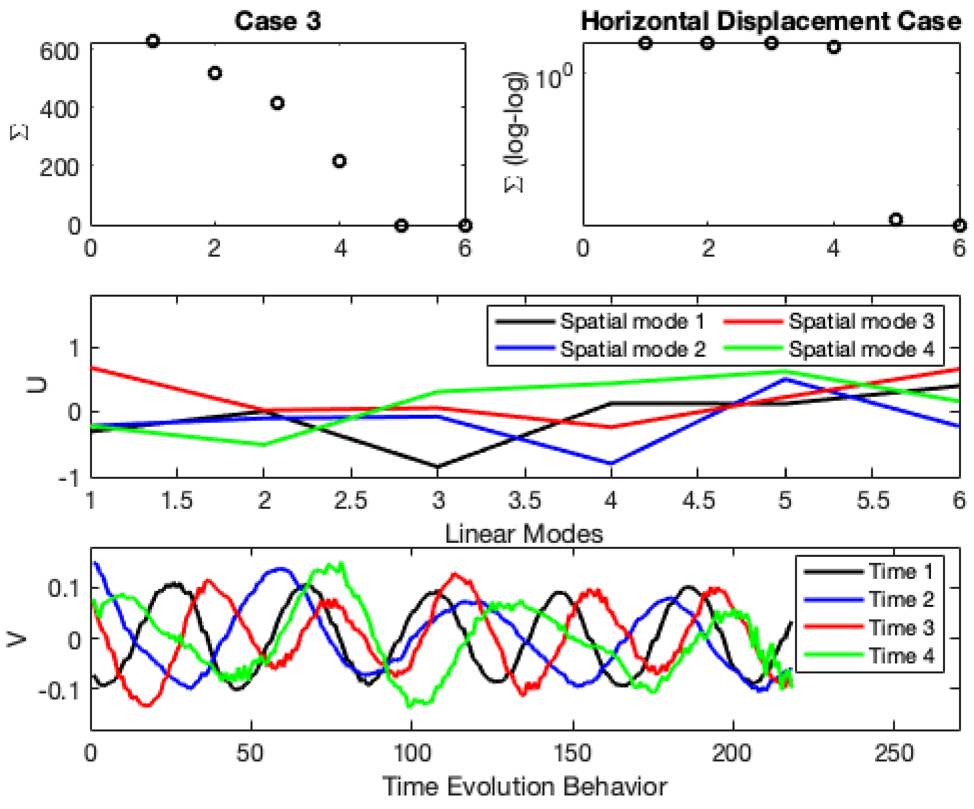
```
load cam3_3.mat

numFrames3_3 = size(vidFrames3_3,4);
for k = 1:numFrames3_3
    mov(k).cdata = vidFrames3_3(:,:,:,k);
    mov(k).colormap = [];
end

for j=1:numFrames3_3
    X3_3=frame2im(mov(j));
    X3_3 = rgb2gray(X3_3); % convert to grayscale
    zoom3_3 = X3_3(180:360,250:500);
    %imshow(zoom3_3>=246); drawnow %; h=gca; h.Visible='On'
    [a,b] = find(zoom3_3>=246); % keep bright pixels
    x3_3(j) = mean(a);
    y3_3(j) = mean(b);
end

[M33 I33] = min(y3_3(1:20));
[ymin3_3] = ind2sub(size(y3_3),I33);

x3_3 = x3_3(ymin3_3:end);
y3_3 = y3_3(ymin3_3:end);

x3_3 = x3_3 - mean(x3_3);
y3_3 = y3_3 - mean(y3_3);
```

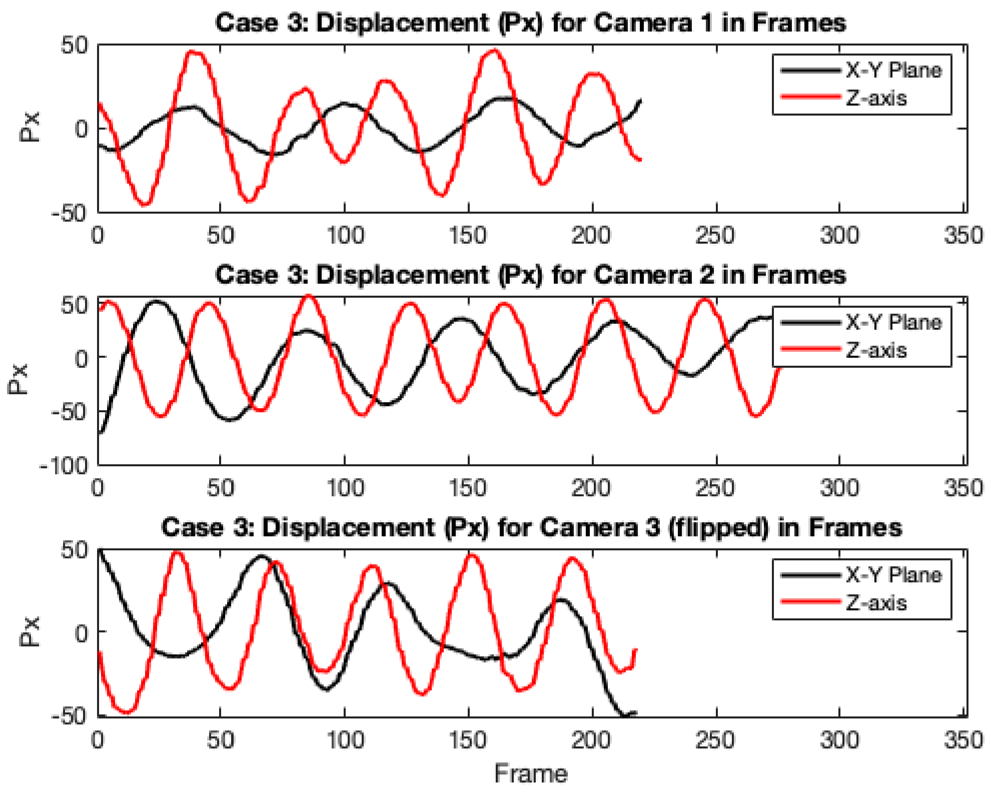
TEST 3: Standardize, PCA, & Plot

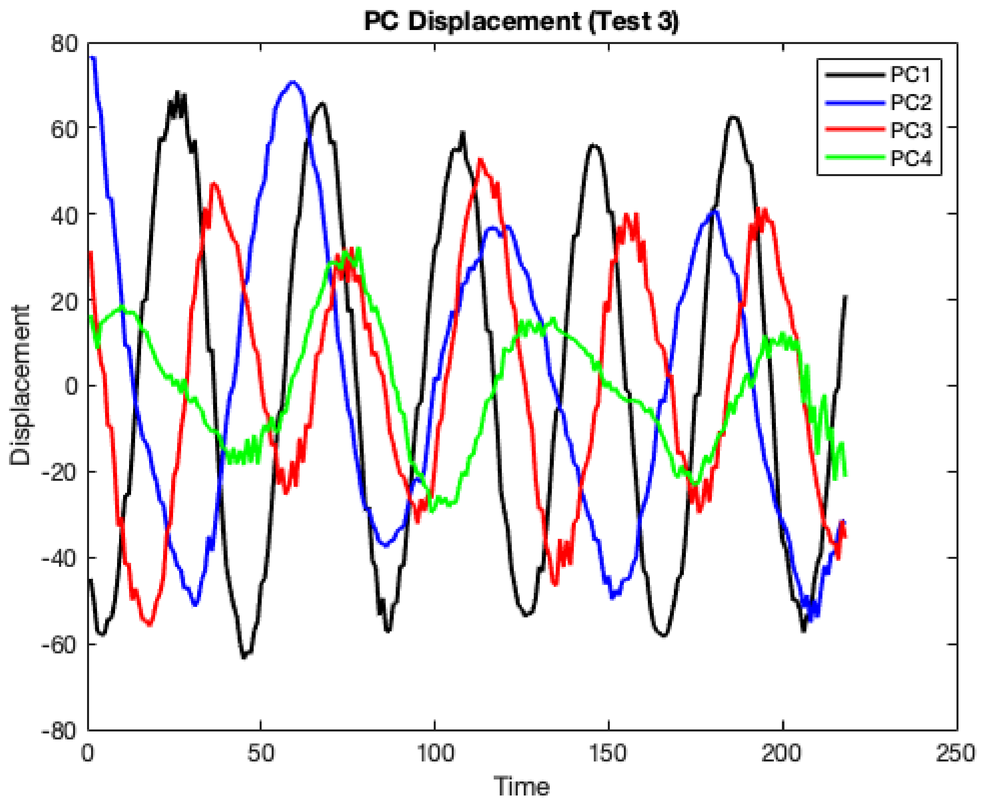
```
if ~isequal(size(x1_3), size(y1_3)) % Make sure each camera x,y size
 match
    disp('Go back to camera1_1')
end
if ~isequal(size(x2_3), size(y2_3))
    disp('Go back to camera2_1')
end
if ~isequal(size(x3_3), size(y3_3))
    disp('Go back to camera3_1')
end

size1_3 = size(y1_3,2); size2_3 = size(y2_3,2); size3_3 =
 size(y3_3,2);
sizer = min([size1_3 size2_3 size3_3]); % take the min length
ind = 1:sizer;
ax = max([size1_3 size2_3 size3_3]);

uu = linspace(1,6,6);
vv = linspace(1,length(ind),length(ind));

X3 = [x1_3(ind); y1_3(ind); x2_3(ind); y2_3(ind); x3_3(ind);
 y3_3(ind)];
[m,n] = size(X3);
mn = mean(X3,2);
X3 = X3 - repmat(mn,1,n);
[u,s,v] = svd(X3);
sigma3 = diag(s);
Y3 = u'*X3;

energy1_3=sigma3(1)/sum(sigma3);
energy2_3=sum(sigma3(1:2))/sum(sigma3);
energy3_3=sum(sigma3(1:3))/sum(sigma3);
energy4_3=sum(sigma3(1:4))/sum(sigma3);
energy5_3=sum(sigma3(1:5))/sum(sigma3);
energy6_3=sum(sigma3(1:6))/sum(sigma3);

% Robust PCA
lambda = 0.2;
```

```
[X3R,X32R] = inexact_alm_rpca(X3.',lambda);
[ur, sr, vr] = svd(X3R.');
sigma3R = diag(sr);
Y3R = ur'*X3R.';
%X1R = X1R'; Y1R = Y1R';

energy1_3R=sigma3R(1)/sum(sigma3R);
energy2_3R=sum(sigma3R(1:2))/sum(sigma3R);
energy3_3R=sum(sigma3R(1:3))/sum(sigma3R);
energy4_3R=sum(sigma3R(1:4))/sum(sigma3R);
energy5_3R=sum(sigma3R(1:5))/sum(sigma3R);
energy6_3R=sum(sigma3R(1:6))/sum(sigma3R);

figure(7)
subplot(3,2,1), plot(sigma3R, 'ko', 'Linewidth', [1.2])
set(gca,'Fontsize',[12]) % energy of each diag variance
%text(3.3,500,{'Singular Values', '(Energy)'}, 'Fontsize',[12])
ylabel('\Sigma'); title('Case 3');

subplot(3,2,2), semilogy(sigma3R,'ko','Linewidth',[1.2])
set(gca,'Fontsize',[12])
%text(3.3,10^(0),'log-log (Energy)','Fontsize',[12])
ylabel('\Sigma (log-log)'); title('Horizontal Displacement Case')

subplot(3,1,2)
plot(uu,ur(:,1),'k',uu,ur(:,2),'b',uu,ur(:,3),'r', ...
    uu,ur(:,4),'g','Linewidth',[2])
set(gca,'Fontsize',[12]); axis([1 6 -1 1.8])
lgd = legend('Spatial mode 1','Spatial mode 2', 'Spatial mode 3', ...
    'Spatial mode 4','Location','northeast');
lgd.NumColumns=2;xlabel('Linear Modes');ylabel('U')

subplot(3,1,3)
plot(vv,vr(:,1),'k',vv,vr(:,2),'b',vv,vr(:,3),'r', ...
    vv,vr(:,4),'g','Linewidth',[2])
set(gca,'Fontsize',[12]); axis([0 270 -0.18 0.18])
legend('Time 1','Time 2', 'Time 3','Time 4','Location','northeast')
xlabel('Time Evolution Behavior');ylabel('V')

figure(8)
subplot(3,1,1)
plot(1:length(y1_3),y1_3,'k',1:length(y1_3),x1_3,'r','Linewidth',[2])
set(gca,'Fontsize',[12]); xlim([0 (ax+70)])
title('Case 3: Displacement (Px) for Camera 1 in Frames')
ylabel('Px'); legend('X-Y Plane','Z-axis')
subplot(3,1,2)
plot(1:length(y2_3),y2_3,'k',1:length(y2_3),x2_3,'r','Linewidth',[2])
set(gca,'Fontsize',[12]); xlim([0 (ax+70)])
title('Case 3: Displacement (Px) for Camera 2 in Frames')
ylabel('Px'); legend('X-Y Plane','Z-axis')
subplot(3,1,3)
plot(1:length(y3_3),x3_3,'k',1:length(y3_3),y3_3,'r','Linewidth',[2])
set(gca,'Fontsize',[12]); xlim([0 (ax+70)])
title('Case 3: Displacement (Px) for Camera 3 (flipped) in Frames')
```

```
xlabel('Frame'); ylabel('Px'); legend('X-Y Plane','Z-axis')

figure(9)
plot(vv,Y3R(1,:),'k',vv,Y3R(2,:),'b', ...
    vv,Y3R(3,:),'r',vv,Y3R(4,:),'g','Linewidth',[2])
set(gca,'Fontsize',[12]),legend('PC1','PC2','PC3','PC4')
xlabel('Time'); ylabel('Displacement'); title('PC Displacement (Test
 3)')

#svd 10 r(A) 4 |E|_0 588 stopCriterion 0.010014
#svd 20 r(A) 4 |E|_0 687 stopCriterion 0.00012237
#svd 30 r(A) 4 |E|_0 722 stopCriterion 1.5606e-06
#svd 37 r(A) 4 |E|_0 723 stopCriterion 9.3219e-08
```




Case 3: Displacement (Px) for Camera 1 in Frames
Case 3: Displacement (Px) for Camera 2 in Frames
Case 3: Displacement (Px) for Camera 3 (flipped) in Frames
X-Y Plane
Z-axis
Px
Frame


PC Displacement (Test 3)
PC1
PC2
PC3
PC4
Displacement
Time

TEST 4: Horizontal Displacement & Rotation, Camera 1

```
load cam1_4.mat

numFrames1_4 = size(vidFrames1_4,4);

for k = 1:numFrames1_4
    mov(k).cdata = vidFrames1_4(:,:,:,k);
    mov(k).colormap = [];
end

for j=1:numFrames1_4
    X1_4=frame2im(mov(j));
    X1_4 = rgb2gray(X1_4); % convert to grayscale
    zoom1_4 = X1_4(180:430,220:430); % zoom around can
    %imshow(zoom1_4>=245); drawnow
    [a,b] = find(zoom1_4>=245);
    x1_4(j) = mean(a);
    y1_4(j) = mean(b);

end

[M14 I14] = min(y1_4(1:20));
[ymin1_4] = ind2sub(size(y1_4),I14); % Find first min to line up

x1_4 = x1_4(ymin1_4:end);
y1_4 = y1_4(ymin1_4:end); % Recast vectors to start at min

x1_4 = x1_4 - mean(x1_4);
y1_4 = y1_4 - mean(y1_4); % Subtract mean to align around 0
```

TEST 4: Horizontal Displacement & Rotation, Camera 2

```
load cam2_4.mat

numFrames2_4 = size(vidFrames2_4,4);
for k = 1:numFrames2_4
    mov(k).cdata = vidFrames2_4(:,:,:,k);
    mov(k).colormap = [];
end

for j=1:numFrames2_4
    X2_4=frame2im(mov(j));
    X2_4 = rgb2gray(X2_4); % convert to grayscale
    zoom2_4 = X2_4(90:400,200:420);
    %imshow(zoom2_4>=250); drawnow %; h=gca; h.Visible='On'
    [a,b] = find(zoom2_4>=250); % keep bright pixels
    x2_4(j) = mean(a);
    y2_4(j) = mean(b);
end

[M24 I24] = min(y2_4(1:20));
[ymin2_4] = ind2sub(size(y2_4),I24);

x2_4 = x2_4(ymin2_4:end);
y2_4 = y2_4(ymin2_4:end);

x2_4 = x2_4 - mean(x2_4);
y2_4 = y2_4 - mean(y2_4);
```

TEST 4: Horizontal Displacement & Rotation, Camera 3

```
load cam3_4.mat

numFrames3_4 = size(vidFrames3_4,4);
for k = 1:numFrames3_4
    mov(k).cdata = vidFrames3_4(:,:,:,k);
    mov(k).colormap = [];
end

for j=1:numFrames3_4
    X3_4=frame2im(mov(j));
    X3_4 = rgb2gray(X3_4); % convert to grayscale
    zoom3_4 = X3_4(150:360,250:500);
    %imshow(zoom3_4>=230); drawnow %; h=gca; h.Visible='On'
    [a,b] = find(zoom3_4>=230); % keep bright pixels
    x3_4(j) = mean(a);
    y3_4(j) = mean(b);
end

[M34 I34] = min(y3_4(1:20));
[ymin3_4] = ind2sub(size(y3_4),I34);

x3_4 = x3_4(ymin3_4:end);
y3_4 = y3_4(ymin3_4:end);

x3_4 = x3_4 - mean(x3_4);
y3_4 = y3_4 - mean(y3_4);
```

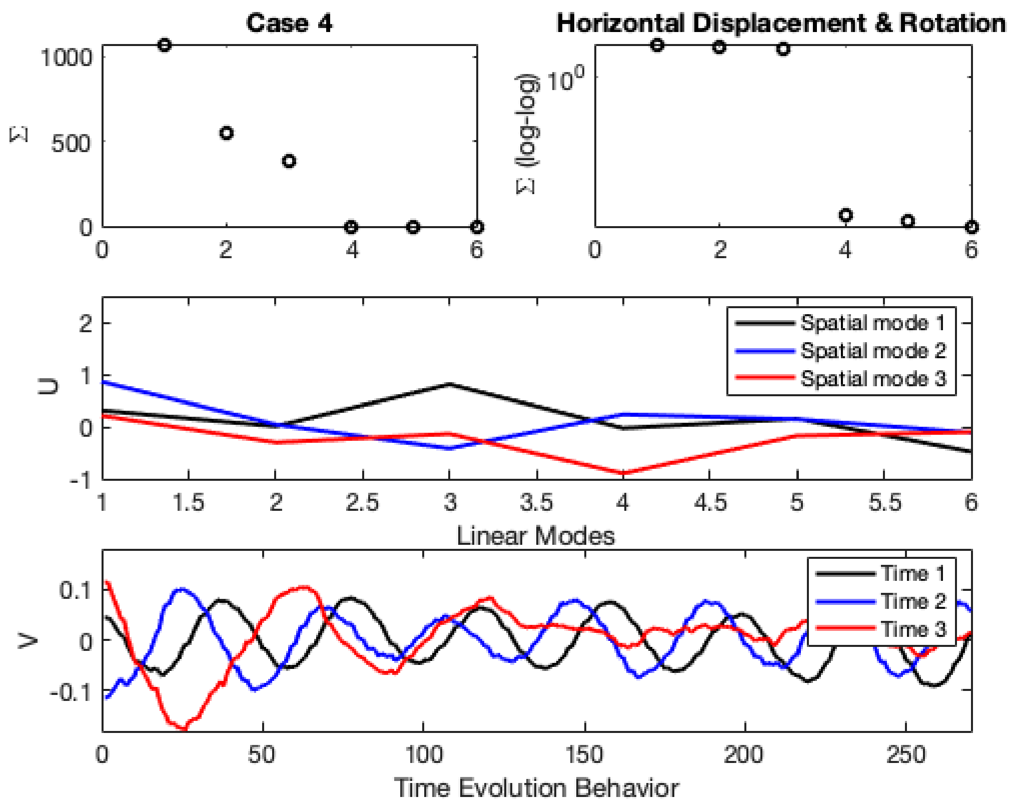
TEST 4: Standardize, PCA, & Plot

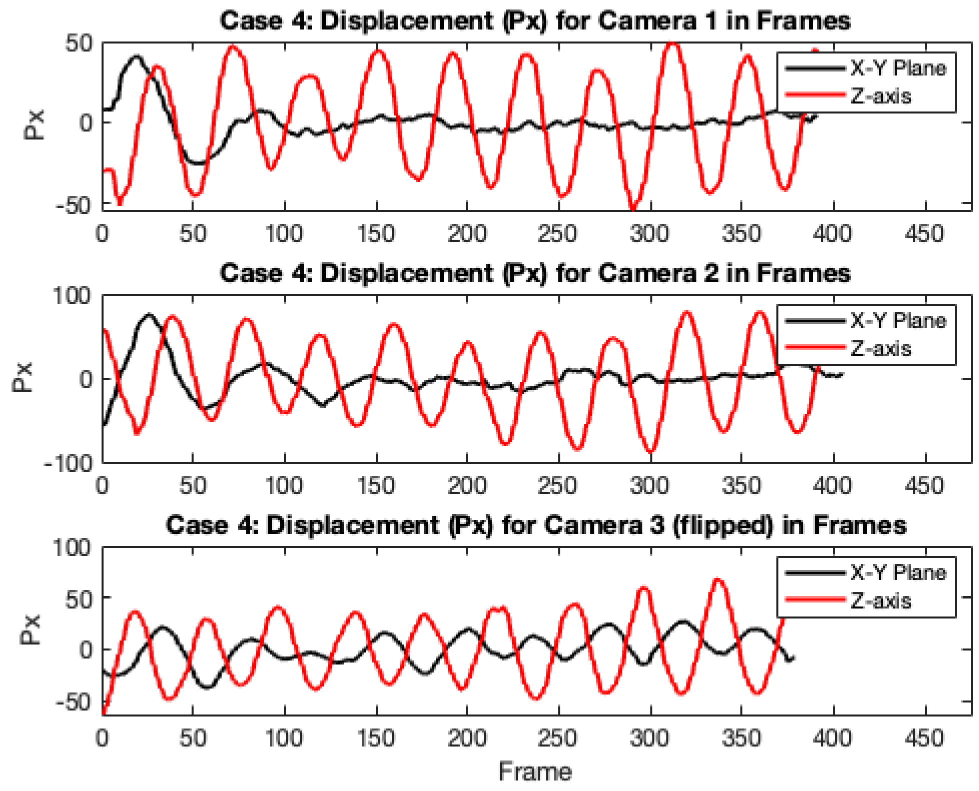
```
if ~isequal(size(x1_4), size(y1_4)) % Make sure each camera x,y size
 match
    disp('Go back to camera1_4')
end
if ~isequal(size(x2_4), size(y2_4))
    disp('Go back to camera2_4')
end
if ~isequal(size(x3_4), size(y3_4))
```

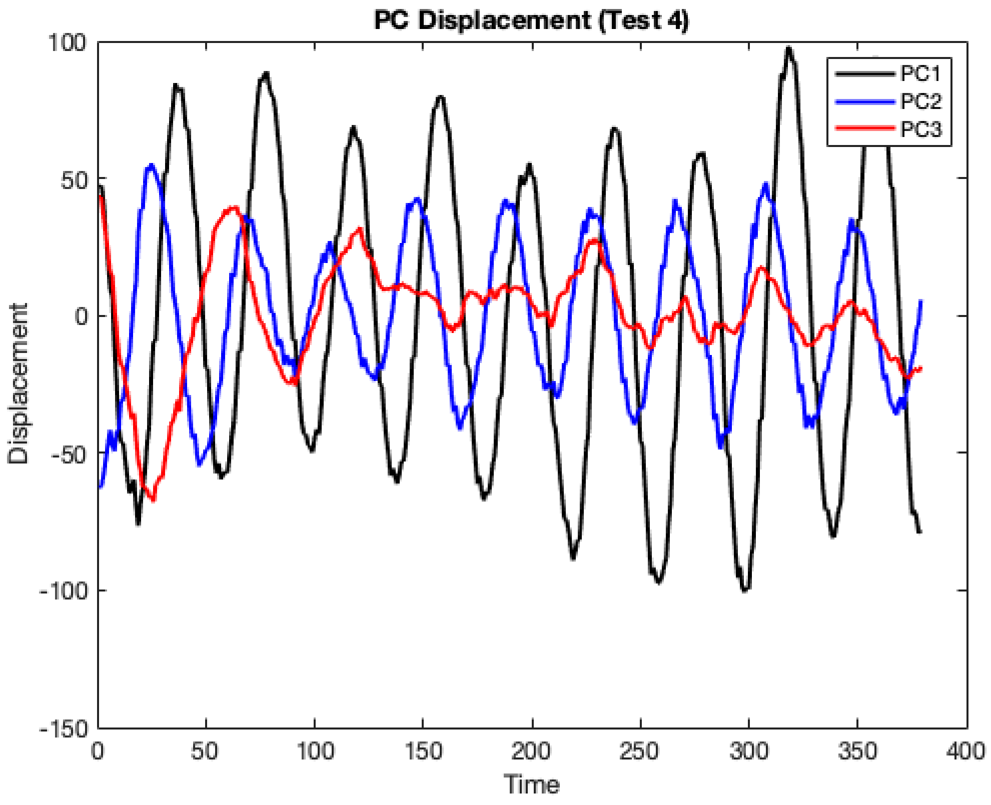
```
    disp('Go back to camera3_4')
end

size1_4 = size(y1_4,2); size2_4 = size(y2_4,2); size3_4 =
 size(y3_4,2);
sizer = min([size1_4 size2_4 size3_4]); % take the min length
ind = 1:sizer;
ax = max([size1_4 size2_4 size3_4]);

uu = linspace(1,6,6);
vv = linspace(1,length(ind),length(ind));

X4 = [x1_4(ind); y1_4(ind); x2_4(ind); y2_4(ind); x3_4(ind);
 y3_4(ind)];
[m,n] = size(X4);
mn = mean(X4,2);
X4 = X4 - repmat(mn,1,n);
[u,s,v] = svd(X4);
sigma4 = diag(s);
Y4 = u'*X4;

energy1_4=sigma4(1)/sum(sigma4);
energy2_4=sum(sigma4(1:2))/sum(sigma4);
energy3_4=sum(sigma4(1:3))/sum(sigma4);
energy4_4=sum(sigma4(1:4))/sum(sigma4);
energy5_4=sum(sigma4(1:5))/sum(sigma4);
energy6_4=sum(sigma4(1:6))/sum(sigma4);

% Robust PCA
lambda = 0.2;
[X4R,X42R] = inexact_alm_rpca(X4.',lambda);
[ur, sr, vr] = svd(X4R.');
sigma4R = diag(sr);
Y4R = ur'*X4R.';
%X1R = X1R'; Y1R = Y1R';

energy1_4R=sigma4R(1)/sum(sigma4R);
energy2_4R=sum(sigma4R(1:2))/sum(sigma4R);
energy3_4R=sum(sigma4R(1:3))/sum(sigma4R);
energy4_4R=sum(sigma4R(1:4))/sum(sigma4R);
energy5_4R=sum(sigma4R(1:5))/sum(sigma4R);
energy6_4R=sum(sigma4R(1:6))/sum(sigma4R);

figure(10)
subplot(3,2,1), plot(sigma4R, 'ko', 'Linewidth', [1.2])
set(gca,'Fontsize',[12]) % energy of each diag variance
%text(3.3,800,{'Singular Values', '(Energy)'}, 'Fontsize',[12])
ylabel('\Sigma'); title('Case 4');

subplot(3,2,2), semilogy(sigma4R,'ko','Linewidth',[1.2])
set(gca,'Fontsize',[12])
%text(3.3,10^(0),'log-log (Energy)','Fontsize',[12])
ylabel('\Sigma (log-log)'); title('Horizontal Displacement &
 Rotation')
```

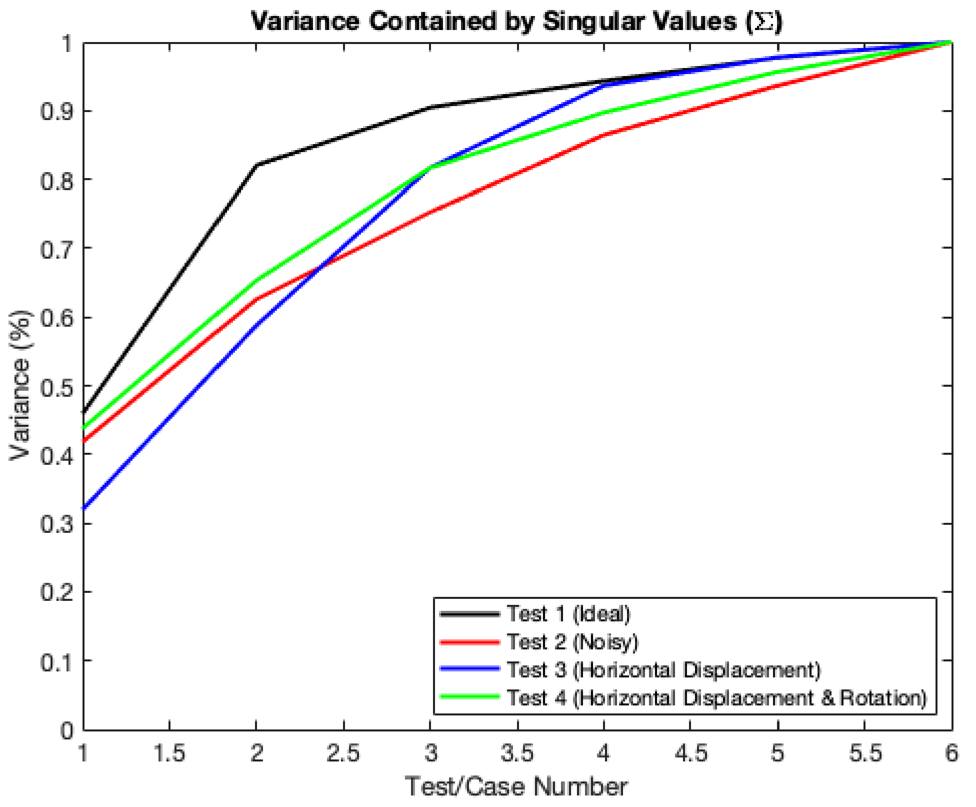
```
subplot(3,1,2)
plot(uu,ur(:,1),'k',uu,ur(:,2),'b',uu,ur(:,3),'r','Linewidth',[2])
set(gca,'Fontsize',[12]); axis([1 6 -1 2.5])
legend('Spatial mode 1','Spatial mode 2', 'Spatial mode 3', ...
    'Location','northeast')
xlabel('Linear Modes');ylabel('U')

subplot(3,1,3)
plot(vv,vr(:,1),'k',vv,vr(:,2),'b',vv,vr(:,3),'r','Linewidth',[2])
set(gca,'Fontsize',[12]); axis([0 270 -0.18 0.18])
legend('Time 1','Time 2', 'Time 3','Location','northeast')
xlabel('Time Evolution Behavior');ylabel('V')

figure(11)
subplot(3,1,1)
plot(1:length(y1_4),y1_4,'k',1:length(y1_4),x1_4,'r','Linewidth',[2])
set(gca,'Fontsize',[12]); xlim([0 (ax+70)])
title('Case 4: Displacement (Px) for Camera 1 in Frames')
ylabel('Px'); legend('X-Y Plane','Z-axis')
subplot(3,1,2)
plot(1:length(y2_4),y2_4,'k',1:length(y2_4),x2_4,'r','Linewidth',[2])
set(gca,'Fontsize',[12]); xlim([0 (ax+70)])
title('Case 4: Displacement (Px) for Camera 2 in Frames')
ylabel('Px'); legend('X-Y Plane','Z-axis')
subplot(3,1,3)
plot(1:length(y3_4),x3_4,'k',1:length(y3_4),y3_4,'r','Linewidth',[2])
set(gca,'Fontsize',[12]); xlim([0 (ax+70)])
title('Case 4: Displacement (Px) for Camera 3 (flipped) in Frames')
xlabel('Frame'); ylabel('Px'); legend('X-Y Plane','Z-axis')

figure(12)
plot(vv,Y4R(1,:),'k',vv,Y4R(2,:),'b',vv,Y4R(3,:),'r','Linewidth',[2])
set(gca,'Fontsize',[12]),legend('PC1','PC2','PC3')
xlabel('Time'); ylabel('Displacement'); title('PC Displacement (Test
 4)')

% figure(13)
% plot(1,energy1_1,'ko',2,energy1_2,'ro',3,energy1_3, ...
%     'bo',4,energy1_4,'go','Linewidth',[2]); hold on
% plot(1,energy3_1,'k+',2,energy3_2,'r+',3,energy3_3, ...
%     'b+',4,energy3_4,'g+','Linewidth',[2]);
% set(gca,'Fontsize',[12]); xlim([0.5 5.5]);
% xlabel('Test/Case Number'); ylabel('Energy');
% title('Comparison of Energy Between Tests');
% legend('Test 1/Energy 1','Test 2/Energy 1', ...
%     'Test 3/Energy 1','Test 4/Energy 1','Test 1/Energy 3', ...
%     'Test 2/Energy 3', 'Test 3/Energy 3', 'Test 4/Energy 3'); hold
 off

figure(14)
t1=[energy1_1 energy2_1 energy3_1 energy4_1 energy5_1 energy6_1];
t2=[energy1_2 energy2_2 energy3_2 energy4_2 energy5_2 energy6_2];
t3=[energy1_3 energy2_3 energy3_3 energy4_3 energy5_3 energy6_3];
```

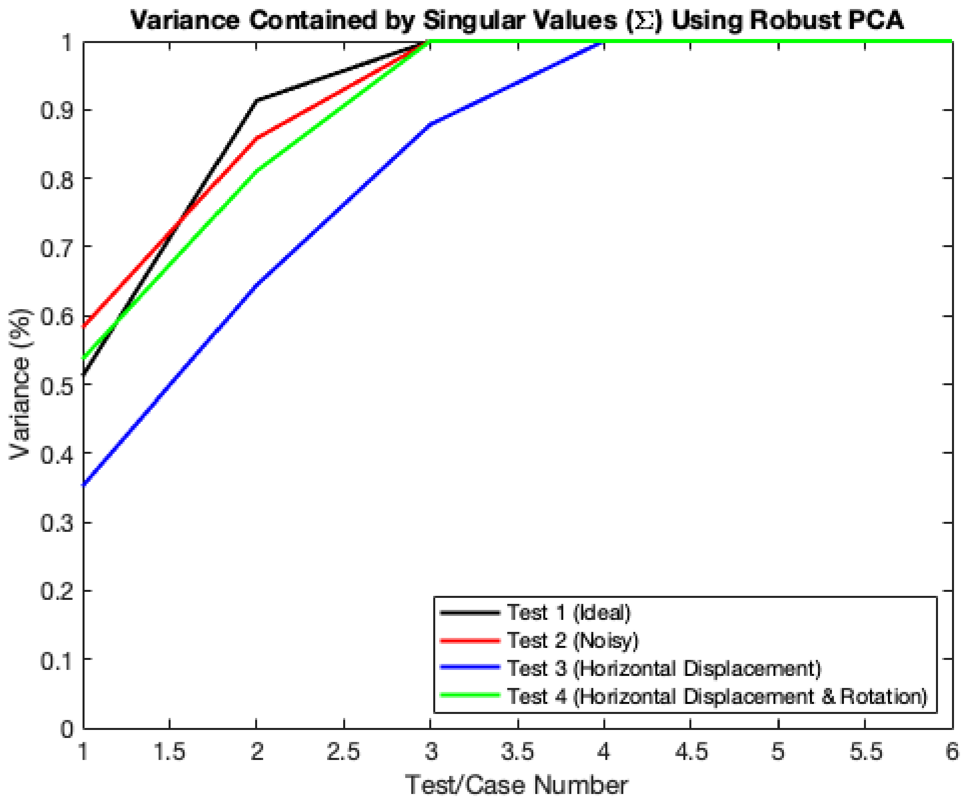
```
t4=[energy1_4 energy2_4 energy3_4 energy4_4 energy5_4 energy6_4];
plot(uu,t1,'k',uu,t2,'r',uu,t3,'b',uu,t4,'g','Linewidth',[2]);
set(gca,'Fontsize',[12]); xlabel('Test/Case Number');
ylabel('Variance (%)'); ylim([0 1]);
legend('Test 1 (Ideal)','Test 2 (Noisy)', ...
    'Test 3 (Horizontal Displacement)', ...
    'Test 4 (Horizontal Displacement &
 Rotation)','Location','southeast');
title('Variance Contained by Singular Values (\Sigma)');

figure(15)
t1R=[energy1_1R energy2_1R energy3_1R energy4_1R energy5_1R
 energy6_1R];
t2R=[energy1_2R energy2_2R energy3_2R energy4_2R energy5_2R
 energy6_2R];
t3R=[energy1_3R energy2_3R energy3_3R energy4_3R energy5_3R
 energy6_3R];
t4R=[energy1_4R energy2_4R energy3_4R energy4_4R energy5_4R
 energy6_4R];
plot(uu,t1R,'k',uu,t2R,'r',uu,t3R,'b',uu,t4R,'g','Linewidth',[2]);
set(gca,'Fontsize',[12]); xlabel('Test/Case Number');
ylabel('Variance (%)'); ylim([0 1]);
legend('Test 1 (Ideal)','Test 2 (Noisy)', ...
    'Test 3 (Horizontal Displacement)', ...
    'Test 4 (Horizontal Displacement &
 Rotation)','Location','southeast');
title('Variance Contained by Singular Values (\Sigma) Using Robust
 PCA');

figure(16)
plot(uu,t1,'k',uu,t2,'r',uu,t3,'b',uu,t4,'g','Linewidth',[2]); hold on
plot(uu,t1R,'k--',uu,t2R,'r--',uu,t3R,'b--',uu,t4R,'g--','Linewidth',
[2]);
set(gca,'Fontsize',[12]); xlabel('Modes');
ylabel('Variance');
legend('Test 1 (Ideal)','Test 2 (Noisy)', ...
    'Test 3 (Horizontal Displacement)', ...
    'Test 4 (Horizontal Displacement & Rotation)', ...
    'Robust Test 1 (Ideal)','Robust Test 2 (Noisy)', ...
    'Robust Test 3 (Horizontal Displacement)', ...
    'Robust Test 4 (Horizontal Displacement & Rotation)',...
    'Location','southeast');
title('Variance Contained Comparison Using PCA & Robust PCA');
 hold off

#svd 10 r(A) 3 |E|_0 939 stopCriterion 0.015204
#svd 20 r(A) 3 |E|_0 1183 stopCriterion 5.4719e-05
#svd 30 r(A) 3 |E|_0 1195 stopCriterion 5.2158e-08
```

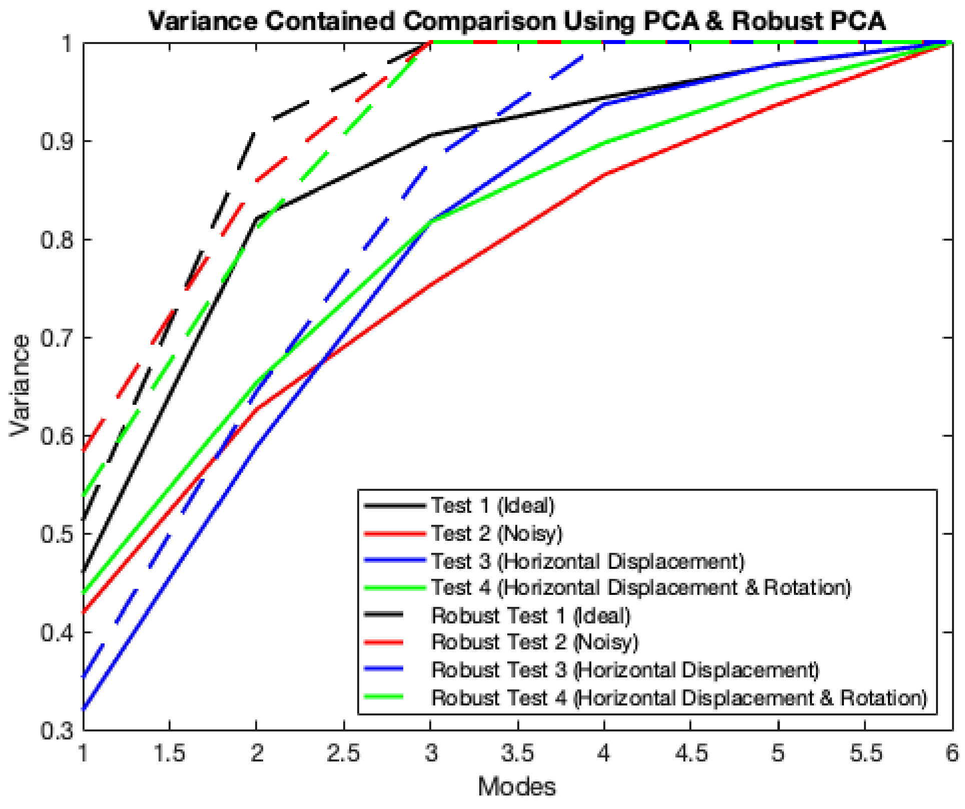

Case 4
Σ
Horizontal Displacement & Rotation
Σ (log-log)
10^0
U
Spatial mode 1
Spatial mode 2
Spatial mode 3
Linear Modes
V
Time 1
Time 2
Time 3
Time Evolution Behavior


Case 4: Displacement (Px) for Camera 1 in Frames
Px
X-Y Plane
Z-axis
Case 4: Displacement (Px) for Camera 2 in Frames
Px
X-Y Plane
Z-axis
Case 4: Displacement (Px) for Camera 3 (flipped) in Frames
Px
X-Y Plane
Z-axis
Frame


PC Displacement (Test 4)
Displacement
Time
PC1
PC2
PC3


Variance Contained by Singular Values (Σ)
Variance (%)
Test/Case Number
Test 1 (Ideal)
Test 2 (Noisy)
Test 3 (Horizontal Displacement)
Test 4 (Horizontal Displacement & Rotation)


Variance Contained by Singular Values (Σ) Using Robust PCA
Variance (%)
Test/Case Number
Test 1 (Ideal)
Test 2 (Noisy)
Test 3 (Horizontal Displacement)
Test 4 (Horizontal Displacement & Rotation)


Variance Contained Comparison Using PCA & Robust PCA
Variance
Modes
Test 1 (Ideal)
Test 2 (Noisy)
Test 3 (Horizontal Displacement)
Test 4 (Horizontal Displacement & Rotation)
Robust Test 1 (Ideal)
Robust Test 2 (Noisy)
Robust Test 3 (Horizontal Displacement)
Robust Test 4 (Horizontal Displacement & Rotation)

Published with MATLAB® R2018b