Principal Component Analysis

Christine M. Baker, GitHub: `cmbaker94`

Feb 24, 2021

Abstract

Principal component analysis is applied to a set of 3 camera images of a oscillatory object for (1) an ideal case, (2) a noisy case, (3) a case with horizontal displacement, and (4) a case with horizontal displacement and rotation. The $x-y$ plane position of the object is identified as the centroid of the image with a threshold and mask applied. Using the singular valued decomposition, the principal component projection is computed and the main modes of oscillations are observed.

1 Introduction and Overview

Principal Component Analysis (PCA) algorithms, specifically single valued decomposition, are applied to movie files created from three different cameras. Four test cases are analyzed to demonstrate the practical usefulness and the effect of noise on the PCA algorithms including (1) an ideal case, (2) a noisy case, (3) a case with horizontal displacement, and (4) a case with horizontal displacement and rotation.

The theoretical background on PCA algorithms are provided in Section 2. The alogrithm implementation and development to resolve the principle components are presented in Section 3. The computational results for the four test cases are shown in Section 4. A conclusions are summarized in Section 5. Details about the MATLAB functions and the function writen for this project are in Appendix A and the MATLAB code to run the analysis is in Appendix B.

2 Theoretical Background

Principal component analysis (PCA) is a method to reduce the dimensionality of large data sets by transforming many variables into a smaller set that still depicts the same information. In particular, PCA can be applied to resolve dominant frequencies from an unknown and low-dimensional system as given in this problem, where a mass oscillates primarily in the z-direction for the ideal case. Given three cameras denoted by subscripts a, b, and c, the data in collected in an arbitrary $x-y$ plane can be extracted from images as:

$$\text{camera1} : (\mathbf{x}_a, \mathbf{y}_a) \tag{1}$$

$$\text{camera2} : (\mathbf{x}_b, \mathbf{y}_b) \tag{2}$$

$$\text{camera1} : (\mathbf{x}_c, \mathbf{y}_c) \tag{3}$$

where the length of each vector $(\mathbf{x}_i, \mathbf{y}_i)$ depends on the length of the time series and the data collection rate. The data can be gathered into a matrix $m \times n$ matrix:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_a \\ \mathbf{y}_a \\ \mathbf{x}_b \\ \mathbf{y}_b \\ \mathbf{x}_c \\ \mathbf{y}_c \end{bmatrix} \tag{4}$$

where m is the number of measurement types (*i.e.,* 2 plans per 3 cameras = 6) and n is the data points in time from a camera. Two main issues with this approach are *noise* and *redundancy.* Noise can ruin the accuracy of the underlying dynamics and can be analyzed by a signal-to-noise ratio. Removing the redundancy of measurements (*i.e.,* 3 cameras measuring a single degree of freedom) is essential for data analysis. The principal components projection (PCP) of the system can be computed with a covariance approach or a single valued decomposition (SVD).

For the covariance approach, the eigenvalues can be extracted from the diagonal of the eigenvalue matrix computed from the covariance matrix, sorted in decreasing order, and based on this order, the eigenvectors are arranged into a vector, V. The principal component projection is computed as the transpose of vector V times the original data matrix X. The covariance matrix is written as

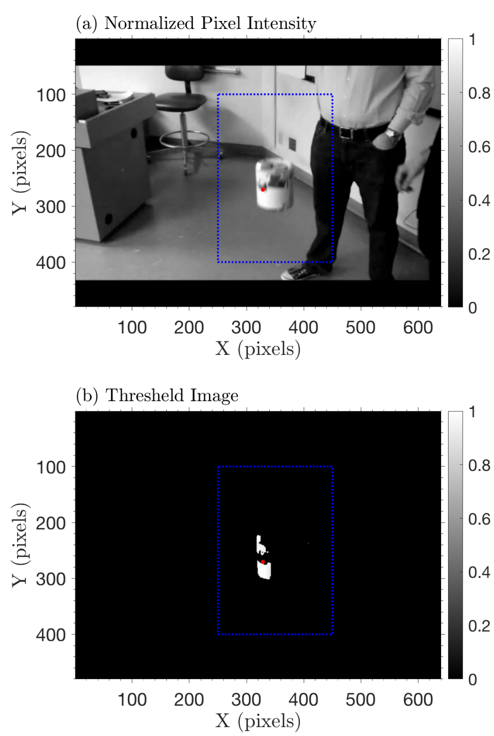

Figure 1: An example from camera 1 test 1 of an image extraction of the center of the paint can mass (red circle). The region of interest (blue dotted line) is shown for the normalized black and white original image (a) and the thresholded image (b).

$$\mathbf{C_X} = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T \tag{5}$$

where $\mathbf{C_X}$ is a square, symmetric $m \times m$ matrix. Large (low) variances correspond to dynamics of interest (non-interesting dynamics) and the off-diagonal terms are the covariance between measurements.

Alternatively, the PCP can be computed with a SVD, a factorization of a matrix into constitutive components or in other words, a stretching/compressing and rotating transformation of vectors into a less redundant system. The factorization known as the *reduced singular value decomposition* can be written as:

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

where $\hat{\mathbf{U}}$ is the $m \times n$ matrix with orthonormal columns, $\hat{\Sigma}$ is the $n \times n$ diagonal matrix, and $\mathbf{V}$ is the $n \times n$ unitary matrix. The full SVD decomposition is

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^* \tag{7}$$

where $\mathbf{U}$ is a $m \times m$ unitary matrix, $\mathbf{V}$ is a $n \times n$ unitary matrix, and Σ is a $m \times n$ diagonal matrix. SVD can diagonalize a matrix with equation 7 by transforming the variable as:

$$\mathbf{Y} = \mathbf{U}^*\mathbf{X} \tag{8}$$

The variance in $\mathbf{Y}$, the transformed variable, is:

$$\mathbf{C_Y} = \frac{1}{n-1}\mathbf{Y}\mathbf{Y}^T = \frac{1}{n-1}\Sigma^2 \tag{9}$$

3 Algorithm Implementation and Development

For each test, the movie files (in matlab files) are loaded into matlab. The camera images were trimmed to the minimum number of frames for the camera with the shortest set of data for each test case. The x, y position of can in each image for each camera is extracted by computing the centroid location of masked and thresholded normalized pixel intensity. To perform this process, the camera image at each time step is read and computed into double precision black and white and normalized by the maximum value. Then, all values below a specific threshold and outside of a boxed region (mask) are set to zero. The selection of the threshold and mask where selected based on visual identification by watching movies of the original image and thresholded image (*e.g.,* Figure 1). The region of interest (masking box) was carefully chosen and adjusted for the three cameras and addition motion of the cameras and the threshold value was typically above 0.9 (Table 1.

The extracted x and y position from each camera is then stored into a matrix X. Once all the cameras are looped through, the single value decomposition (SVD) is computed from the X matrix of data size $m \times n$ with the mean of each row removed divided by the square root of $n-1$. The SVD method is applied to the camera data since it is more robust and suggested to be used (Kutz, 2013). Then the principal components projection is computed as the transpose of the U matrix from the SVD function times the X matrix. The variance for each mode is extracted as the squared of the diagonals from the Σ in the SVD output. This process is completed for all four trials. Then, a plot of the first three modes for each case is created.

Additionally, the camera time start offset and the subsequent impact on the PCA modes was investigated by trimming the start time of cameras based on visual identification in images (*e.g.,* Figure 1a).

Table 1: The threshold value and region of interest ([y_{min} y_{max} x_{min} x_{max}]) for each camera for each test case.

Test	camera	region of interest	threshold
1	1	[100 420 250 450]	0.9
1	2	[50 435 200 380]	0.94
1	3	[200 350 200 500]	0.84
2	1	[150 400 250 450]	0.92
2	2	[50 435 200 420]	0.94
2	3	[200 350 200 500]	0.9
3	1	[150 400 250 450]	0.92
3	2	[175 445 100 500]	0.94
3	3	[200 350 200 500]	0.9
4	1	[150 400 250 500]	0.92
4	2	[50 435 200 500]	0.96
4	3	[100 350 200 500]	0.9

4 Computational Results

The PCA algorithm was applied to four test cases: (1) an ideal case, (2) a noisy case, (3) a case with horizontal displacement, and (4) a case with horizontal displacement and rotation. The first four modes, where most of the variance is captured in modes 1-3, are presented (Figure 2, 3).

For **test 1**, mode 1 and 2 show oscillation at a similar frequency. The offset is likely due to the difference in the time when the cameras began capturing images. Oscillations of the mode 3 and subsequent modes are small, because the idealized test primarily only has motion along one axis, so the dynamics can be easily represented with less modes. The variances in mode 1 are much greater than mode 2 and lower, indicating that most of the variance is represented by the first mode.

The sensitivity of the PCA modes to the camera time offset is investigated. When the camera offset is minimized the variance in the time series is shifted from mode 2 to 1, as demonstrated by the larger (smaller) oscillatory amplitudes in mode 1 (2) (Figure 2). If the cameras were started at exactly the same time, it is expected that most of the variance would be in mode 1 with even less in mode 2 and greater.

For **test 2**, the output is much noisier (less smooth) due to the motion of the camera, especially camera 2. The PCA algorithm is able to isolate the oscillations in the first mode; however, the sinusoidal motion is not as smooth and the amplitude varies with time. The oscillations at mode 2 and great are much smaller and also noisy. Although this case is still mainly only oscillations in the z-direction, due to the noise, the variance in the first mode is smaller than test 1. This test has the highest variance at the largest modes (5,6) which is likely due to the noise from the camera motion.

For **test 3**, the first component is the vertical oscillation of the object, which can be observed via the similar frequency to test 1. The second and third mode represent the horizontal motion in two planes, x and y. The variance for the first three modes are relatively similar and decreasing with mode, while mode 4-6 are much smaller. This indicates that most of the variance is in the first 3 modes. The higher modes have longer oscillations likely associated with the horizontal motion.

For **test 4**, the fluctuations are more similar to test 3 with vertical oscillations of the object

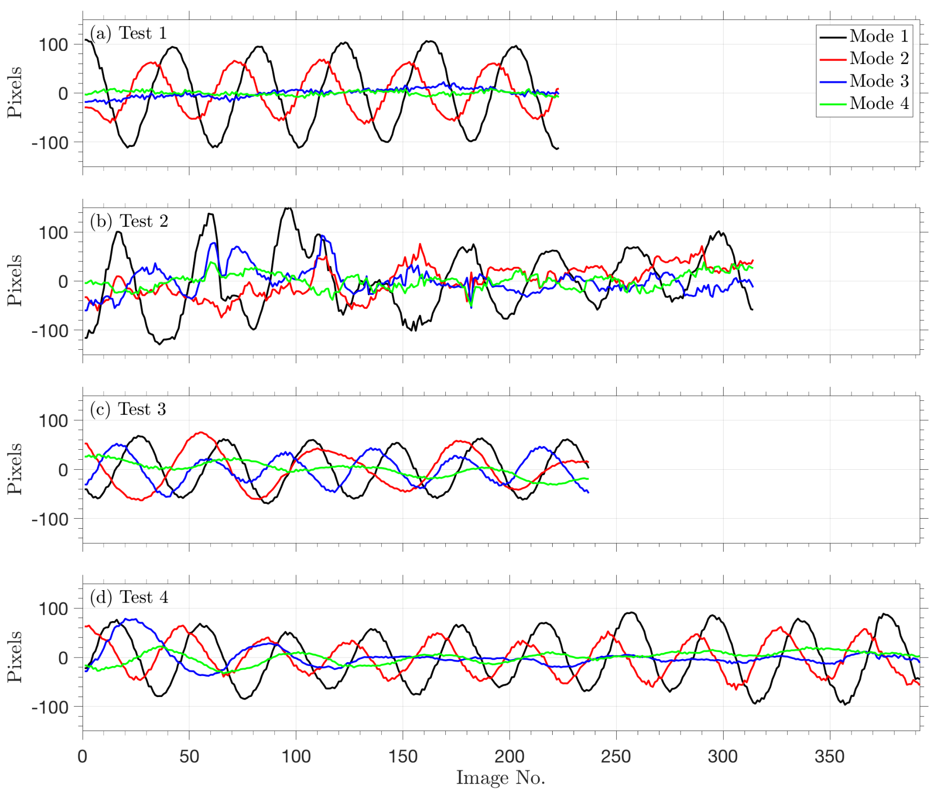

Figure 2: The first 4 modes of the principal component projections for the four test cases: (a) an ideal case, (b) a noisy case, (c) a case with horizontal displacement, and (d) a case with horizontal displacement and rotation.

and the horizontal motions are likely represented in mainly in mode 2 because the oscillations in the third mode vary in frequency and amplitude from mode 2. Mode 3 or 4 may represent the rotational motion.

The initial variance for tests 3 and 4 are much smaller than the other cases, which may be due to additional variance in the horizontal direction (Figure 3). The first mode for all tests oscillate at a similar frequency which is associated with the spring constant. These values could be used to find the frequency and amplitude associated with a spring mass system (Kutz, 2013).

5 Summary and Conclusions

Principal component analysis is applied to a set of 3 camera images at different angles of a oscillatory paint can on a string for four test cases: (1) an ideal case, (2) a noisy case, (3) a case with horizontal displacement, and (4) a case with horizontal displacement and rotation. To extract the x, y position of the paint can in each image, a threshold and mask was applied to the normalized black and white image. The singular valued decomposition method was used to resolve the the principal component projection based on the positions extracted from the camera. For test 1 and 2, most of the variance is described in the first mode and associated with vertical oscillations. Horizontal motion is in the second and possibly third mode for test 3 and 4. This PCA analysis could be used to resolve the physics of this spring mass scenario.

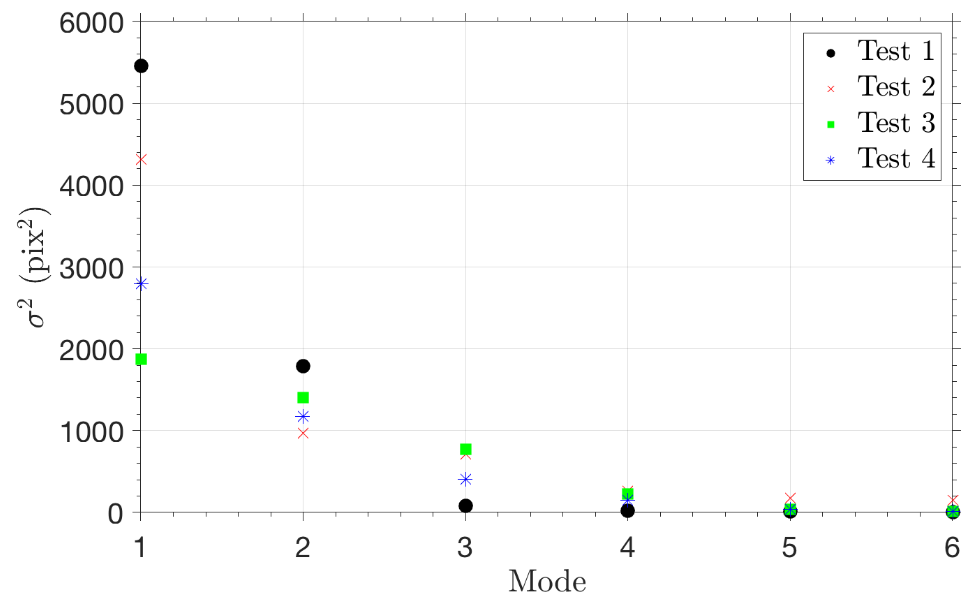

Figure 3: The variance (x-axis) for each mode (y-axis, modes 1-6) for the four test cases: (a) an ideal case, (b) a noisy case, (c) a case with horizontal displacement, and (d) a case with horizontal displacement and rotation.

Appendix

A MATLAB functions used and brief implementation explanation

- `axis`: indicate the limits for the current plotted axes
- `calc_pcp`: see below
- `figure`: open new figure
- `diag`: extract diagonals
- `get_xy_thresh`: see below
- `grid`: add grid lines to a figure
- `*label`: label the x, y, z axes of plots
- `length`: find the length of the largest array dimension (grab the time-dimension in loops)
- `load`: load data from a .m file into workspace (load subdata.m)
- `max`: find the maximum value of an array or matrix
- `mean`: compute mean
- `min`: find the minimum value of an array or matrix
- `pcolor`: plotting contour function
- `print`: export and save figure
- `regionprops`: compute weighted centroid of the image
- `repmat`: created matrix with repeated values
- `scatter`: create scatter plot
- `set`: manipulate a figure
- `size`: grab size of a matrix
- `sqrt`: compute square root
- `svd`: compute singular value decomposition values
- `zeros`: create a matrix of zeros
- `text`: define text for a figure

Function to extract x,y vectors:

```
function [x,y] = get_xy_thresh(cam,thresh,ROI,trim,pltfig,ffcam)
% Extract the x y vectors of positions from the images
% INPUT:
% cam: camera 4d matrix
% thresh: threshold value
% ROI: region of interest
% trim: number of images to trip to
% pltfig: flag if plotting
% ffcam: fig folder camera, test
% OUTPUT:
% x: x plane vector
% y: y plane vector

if pltfig == 1
    figure('units','inches','position',[1 1 8 12],'Color','w');
    eval(['!mkdir ',ffcam])
end

[height width rgb num_frames] = size(cam);

for j=1:trim
    X=cam(:,:,:,j); % extract camera image at each times step
    Xbw = double(rgb2gray(X)); % convert to bw
    Xback = Xbw/max(Xbw,[],'all'); % store normalized image
    Xnorm = Xbw/max(Xbw,[],'all'); % create normalized image to max
    Xnorm(Xnorm<thresh)=0; % threshold image
    xmask = zeros(size(Xnorm)); % create mask matrix
    xmask(ROI(1):ROI(2),ROI(3):ROI(4)) = 1; % create mask
    Xnorm = Xnorm.*xmask; % multiply by mask
    props = regionprops(true(size(Xnorm)), Xnorm, 'WeightedCentroid'); % find centroid
    x(j)  = props.WeightedCentroid(1);
    y(j) = props.WeightedCentroid(2);
    if j < trim
        if pltfig == 1
            subplot(2,1,1)
            pcolor(Xback); shading interp; colorbar;
            hold on
            plot([ROI(3) ROI(3) ROI(4) ROI(4) ROI(3)],[ROI(1) ROI(2) ROI(2) ROI(1) ROI(1)],'b','LineWidth',2,'LineStyle',':')
            scatter(props.WeightedCentroid(1),props.WeightedCentroid(2),20,'r','fill');
            colormap('gray')
            set(gca, 'YDir','reverse')
            text(0,-25,'(a) Normalized Pixel Intensity','interpreter'
```

```
        ,'latex','fontsize',20,'Color',[0 0 0]);
            xlabel('X (pixels)','interpreter','latex','fontsize',20)
            ylabel('Y (pixels)','interpreter','latex','fontsize',20)
            grid on
            box on
            h1=gca;
            set(h1,'fontsize',20);
            set(h1,'tickdir','out','xminortick','on','yminortick','on');
            set(h1,'ticklength',1*get(h1,'ticklength'));

            subplot(2,1,2)
            pcolor(Xnorm); shading interp; colorbar;
            hold on
            plot([ROI(3) ROI(3) ROI(4) ROI(4) ROI(3)],[ROI(1) ROI(2) ROI(2) ROI(1) ROI(1)],'b','LineWidth',2,'LineStyle',':')
            scatter(props.WeightedCentroid(1),props.WeightedCentroid(2),20,'r','fill');
            colormap('gray')
            set(gca, 'YDir','reverse')
            set(gca, 'YDir','reverse')
            text(0,-25,'(b) Thresheld Image','interpreter','latex','fontsize',20,'Color',[0 0 0]);
            xlabel('X (pixels)','interpreter','latex','fontsize',20)
            ylabel('Y (pixels)','interpreter','latex','fontsize',20)
            grid on
            box on
            h1=gca;
            set(h1,'fontsize',20);
            set(h1,'tickdir','out','xminortick','on','yminortick','on');
            set(h1,'ticklength',1*get(h1,'ticklength'));
            drawnow
            Sname = [ffcam,'/example_',num2str(j)];
            print(Sname,'-dpng')
            clf
        end
    end
end

end
```

Function to compute principal component projection:

```
function [Y,svdout] = calc_pcp(X)
% compute the principal component projection
% INPUT: X matrix of data valeus
```

```
% OUTPUT: Y: PCP matrix, svdout: outputs from SVD function

[m,n]=size(X); % compute data size
mn=mean(X,2); % compute mean for each row
X=X-repmat(mn,1,n); % subtract mean

[u,s,v]=svd(X/sqrt(n-1)); % perform the SVD
lambda=diag(s).^2; % produce diagonal variances
Y=u'*X; % produce the principal components projection
svdout.u = u;
svdout.s = s;
svdout.v = v;
svdout.lambda = lambda;
```

MATLAB code

Code avaialble at: `https://github.com/cmbaker94/Baker`$_{A}MATH582$

```
clear all
close all
clc

addpath(genpath('/Users/cmbaker9/Documents/MTOOLS'))

%% STEP 0: Locate and load data

datapath    = '/Users/cmbaker9/Documents/UW_Classes/
   AMATH_Data_Analysis/HW3/data/';
figfolder   = '/Users/cmbaker9/Documents/UW_Classes/
   AMATH_Data_Analysis/HW3/figures/';

%% Test 1: Ideal Case

numcam = 3;
matchcam = 1;
% load data
load([datapath,'cam1_1.mat']);
load([datapath,'cam2_1.mat']);
load([datapath,'cam3_1.mat']);

A = [];
pltfig = 0; % 1 if plot theshold figure

thresh = [0.9 0.94 0.84]; % threshold value for each camera
ROI = [100 420 250 450;...
    50 435 200 390;...
```

```
    200 350 200 500]; % region of interest for each camera [ymin ymax xmin xmax]

if matchcam == 1
    vidFrames2_1 = squeeze(vidFrames2_1(:,:,:,19:end));
    vidFrames3_1 = squeeze(vidFrames3_1(:,:,:,10:end));
    trim = size(vidFrames3_1,4); % minimum number of frames to trim to
else
    trim = size(vidFrames1_1,4); % minimum number of frames to trim to
end


for i = 1:numcam
    eval(['cam = vidFrames',num2str(i),'_1;']) % camera of interest
    ffcam = [figfolder,'T1C',num2str(i)];
    [x,y] = get_xy_thresh(cam,thresh(i),ROI(i,:),trim,pltfig,ffcam); % extract xy
    A = [A; x; y];
end

[Y1,svd1] = calc_pcp(A); % compute prinicipal component projection with svd

if matchcam == 1
    figure('Color','w')
    plot(Y1(1,:),'Color','k','LineWidth',2)
    hold on
    plot(Y1(2,:),'Color','r','LineWidth',2)
    plot(Y1(3,:),'Color','b','LineWidth',2)
    ylabel('Pixels','interpreter','latex','fontsize',20)
    xlabel('Image No.','interpreter','latex','fontsize',20)
    grid on
    box on
    h1=gca;
    set(h1,'fontsize',20);
    set(h1,'tickdir','out','xminortick','on','yminortick','on');
    set(h1,'ticklength',1*get(h1,'ticklength'));

    Sname = [figfolder,'match_cam_T1'];
    print(Sname,'-dpng')
end

clear vidFrames* cam ROI tresh A
```

```
%% Test 2: Noisy Case
% see comments in test 1

numcam = 3;
load([datapath,'cam1_2.mat']);
load([datapath,'cam2_2.mat']);
load([datapath,'cam3_2.mat']);

A = [];
pltfig = 0;

thresh = [0.92 0.94 0.9];
ROI = [150 400 250 450;...
    50 435 200 420;...
    200 350 200 500];
trim = size(vidFrames1_2,4);

for i = 1:numcam
    eval(['cam = vidFrames',num2str(i),'_2;'])
    ffcam = [figfolder,'T2C',num2str(i)];
    [x,y] = get_xy_thresh(cam,thresh(i),ROI(i,:),trim,pltfig,ffcam);
    A = [A; x; y];
end

[Y2,svd2] = calc_pcp(A);

clear vidFrames* cam ROI tresh A

%% Test 3: Horizontal Displacement
% see comments in test 1

numcam = 3;
load([datapath,'cam1_3.mat']);
load([datapath,'cam2_3.mat']);
load([datapath,'cam3_3.mat']);

A = [];
pltfig =0;

thresh = [0.92 0.94 0.9];
ROI = [150 400 250 450;...
    175 445 100 500;...
    200 350 200 500];
trim = size(vidFrames3_3,4);
```

```
for i = 1:numcam
    eval(['cam = vidFrames',num2str(i),'_3;'])
    ffcam = [figfolder,'T3C',num2str(i)];
    [x,y] = get_xy_thresh(cam,thresh(i),ROI(i,:),trim,pltfig,ffcam
       );
    A = [A; x; y];
end

[Y3,svd3] = calc_pcp(A);

clear vidFrames* cam ROI tresh A

%% Test 4: Horizontal Displacement and Rotation
% see comments in test 1

numcam = 3;
load([datapath,'cam1_4.mat']);
load([datapath,'cam2_4.mat']);
load([datapath,'cam3_4.mat']);

A = [];
pltfig = 0;

thresh = [0.92 0.96 0.9];
ROI = [150 400 250 500;...
    50 435 200 500;...
    100 350 200 500];
trim = size(vidFrames1_4,4);

for i = 1:numcam
    eval(['cam = vidFrames',num2str(i),'_4;'])
    ffcam = [figfolder,'T4C',num2str(i)];
    [x,y] = get_xy_thresh(cam,thresh(i),ROI(i,:),trim,pltfig,ffcam
       );
    A = [A; x; y];
end

[Y4,svd4] = calc_pcp(A);

clear vidFrames* cam ROI tresh A

%% Create Plot
xreg = [0 392];
yreg = [-150 150];
```

```
figure('units','inches','position',[1 1 16 16],'Color','w');

clf

ax1 = axes('Position',[0.12 0.77 0.8 0.18]);
tvec = 1:size(Y1,2);
plot(ax1,tvec,Y1(1,:),'Color','k','LineWidth',2)
hold on
plot(ax1,tvec,Y1(2,:),'Color','r','LineWidth',2)
plot(ax1,tvec,Y1(3,:),'Color','b','LineWidth',2)
plot(ax1,tvec,Y1(4,:),'Color','g','LineWidth',2)
text(3,120,'(a) Test 1','interpreter','latex','fontsize',20,'Color',[0 0 0]);
ylabel('Pixels','interpreter','latex','fontsize',20)
xlim(xreg)
ylim([-150 150])
grid on
box on
h1=gca;
set(h1,'fontsize',20);
set(h1,'tickdir','out','xminortick','on','yminortick','on');
set(h1,'ticklength',1*get(h1,'ticklength'));
set(h1,'xtick',[0:50:400],'xticklabel',{'' '' '' ''});
h2 = legend('Mode 1','Mode 2','Mode 3','Mode 4');
set(h2,'interpreter','latex','fontsize',20,'orientation','vertical','Location','northeast');

ax2 = axes('Position',[0.12 0.54 0.8 0.18]);
tvec = 1:size(Y2,2);
plot(ax2,tvec,Y2(1,:),'Color','k','LineWidth',2)
hold on
plot(ax2,tvec,Y2(2,:),'Color','r','LineWidth',2)
plot(ax2,tvec,Y2(3,:),'Color','b','LineWidth',2)
plot(ax2,tvec,Y2(4,:),'Color','g','LineWidth',2)
text(3,120,'(b) Test 2','interpreter','latex','fontsize',20,'Color',[0 0 0]);
ylabel('Pixels','interpreter','latex','fontsize',20)
xlim(xreg)
ylim([-150 150])
grid on
box on
h1=gca;
set(h1,'fontsize',20);
set(h1,'tickdir','out','xminortick','on','yminortick','on');
set(h1,'ticklength',1*get(h1,'ticklength'));
set(h1,'xtick',[0:50:400],'xticklabel',{'' '' '' ''});
```

```
ax3 = axes('Position',[0.12 0.31 0.8 0.18]);
tvec = 1:size(Y3,2);
plot(ax3,tvec,Y3(1,:),'Color','k','LineWidth',2)
hold on
plot(ax3,tvec,Y3(2,:),'Color','r','LineWidth',2)
plot(ax3,tvec,Y3(3,:),'Color','b','LineWidth',2)
plot(ax3,tvec,Y3(4,:),'Color','g','LineWidth',2)
text(3,120,'(c) Test 3','interpreter','latex','fontsize',20,'Color'
    ',[0 0 0]);
ylabel('Pixels','interpreter','latex','fontsize',20)
xlim(xreg)
ylim(yreg)
grid on
box on
h1=gca;
set(h1,'fontsize',20);
set(h1,'tickdir','out','xminortick','on','yminortick','on');
set(h1,'ticklength',1*get(h1,'ticklength'));
set(h1,'xtick',[0:50:400],'xticklabel',{'' '' '' ''});

ax4 = axes('Position',[0.12 0.08 0.8 0.18]);
tvec = 1:size(Y4,2);
plot(ax4,tvec,Y4(1,:),'Color','k','LineWidth',2)
hold on
plot(ax4,tvec,Y4(2,:),'Color','r','LineWidth',2)
plot(ax4,tvec,Y4(3,:),'Color','b','LineWidth',2)
plot(ax4,tvec,Y4(4,:),'Color','g','LineWidth',2)
text(3,120,'(d) Test 4','interpreter','latex','fontsize',20,'Color'
    ',[0 0 0]);
ylabel('Pixels','interpreter','latex','fontsize',20)
xlabel('Image No.','interpreter','latex','fontsize',20)
xlim(xreg)
ylim(yreg)
grid on
box on
h1=gca;
set(h1,'fontsize',20);
set(h1,'tickdir','out','xminortick','on','yminortick','on');
set(h1,'ticklength',1*get(h1,'ticklength'));
set(h1,'xtick',[0:50:400],'xticklabel',{'0' '50' '100' '150' '200'
    '250' '300' '350' '400'});

Sname = [figfolder,'results'];
print(Sname,'-dpng')
```

```
%% Scatter

figure('units','inches','position',[1 1 10 6],'Color','w');
mode = 1:6;
scatter(mode,svd1.lambda,100,'k','fill')
hold on
scatter(mode,svd2.lambda,100,'r','x')
scatter(mode,svd3.lambda,100,'g','sq','fill')
scatter(mode,svd4.lambda,100,'b','*')
% text(3,120,'(a) Test 1','interpreter','latex','fontsize',20,'
    Color',[0 0 0]);
xlabel('Mode','interpreter','latex','fontsize',20)
ylabel('$\sigma^2$ (pix$^2$)','interpreter','latex','fontsize',20)
xlim([1 6])
% ylim(yreg)
grid on
box on
h1=gca;
set(h1,'fontsize',20);
set(h1,'tickdir','out','xminortick','on','yminortick','on');
set(h1,'ticklength',1*get(h1,'ticklength'));
h2 = legend('Test 1','Test 2','Test 3','Test 4');
set(h2,'interpreter','latex','fontsize',20,'orientation','vertical
    ','Location','northeast');

Sname = [figfolder,'variance'];
print(Sname,'-dpng')
```

References

J Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data.* Oxford University Press, 2013.