# AMATH 582 HW 3

James Pratt

21 Feb 2020

**Abstract**

Singular Value Decomposition (SVD) and Principal Component Analysis (PCA) enabled me to successfully extract the motion of a paint can from videos taken from three different cameras in three different orientations, for each of four different cases. For all cases, the primary motion of the paint can was vertical; the degree of motion in horizontal directions varied depending on the case. The motion of the paint can was extracted by tracking maximum pixel values in each frame, converting the indices of those pixels into $x$ and $y$ coordinates, and using a matrix of those coordinates as input to the SVD/PCA, which output matrices of the PCA mode directions, the paint can's trajectory, and singular values for each PCA mode.

## 1 Introduction and Overview

The goal of this assignment was to track the motion of a paint can, on what appeared to be an elastic string, from video recordings taken with three different cameras in three different locations and orientations with respect to the paint can. The paint can had a flashlight attached to its top; the brightness of this flashlight served as the basis for the tracking algorithm implemented to extract $x$ and $y$ coordinates of the paint can's motion for each video frame.

Four cases were analyzed for this assignment: in case 1, the paint can's motion was almost exclusively in the vertical direction, with very little horizontal or rotational components. Case 2 was largely identical, except noise was introduced into each video in the form of camera shake. In case 3, the paint can was released slightly off center to introduce some horizontal components to its trajectory, in addition to the vertical motion. Case 4 introduced a rotational component to the motion, resulting in increased difficulty in tracking the flashlight at the top of the paint can.

In short, the paint can's trajectory was extracted from each video by finding the maximum intensity pixel in each video frame, converting its indices to $x$ and $y$ coordinates, and building vectors of these coordinates into a matrix. Singular Value Decomposition (SVD) was applied to this matrix using MATLAB's built-in function. The first three modes of the Principal Component Analysis (PCA) were then plotted along with the principal values themselves to view the predominant modes in the paint can's motion. The algorithm and results are examined in detail in the following sections.

## 2 Theoretical Background

This assignment primarily required the use of Singular Value Decomposition (SVD) and Principal Component Analysis (PCA), the full details of which may be found in our textbook [1]. For the purposes of this assignment, SVD was used to extract coordinates of the paint can's trajectory, and PCA was used to determine the predominant modes of the trajectory.

In order to use the SVD method, each camera's two-dimensional representation of the paint can's trajectory had to be arranged into a matrix as follows:

$$X = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \end{bmatrix} \tag{1}$$

where each subscript indicates the camera number. The matrix $X \in \mathbb{R}^{m \times n}$ contains $m$ measurement types (in this case, $m = 6$ for six measurement types, two from each camera) and $n$ data points or observations (in this case, the number of frames contained in each video).

Since each camera recorded the same event, the data produced by each camera are redundant, in the sense that they are statistically dependent. Part of the SVD involves computing the covariance matrix (denoted by $S$ for this assignment), a square, symmetric $m \times m$ matrix (where $m = 6$ for our case) whose diagonal entries represent the variance of each particular measurement set, and whose off-diagonal entries represent the covariances between data sets. Large off-diagonal terms indicate high statistical correlation between two data sets, where small off-diagonal terms indicate low redundancy and statistical independence between data sets.

The covariance matrix is computed as follows:

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

where $X$ is the matrix containing our experimental data, and $n$ is the number of data points taken by each camera. Note that the covariance matrix is also a square, symmetric $m \times m$ matrix. In addition, large diagonal terms generally represent the dynamics of interest; in our case, the motion of the paint can. Low variances correspond to uninteresting dynamics.

In order to extract the fundamental or essential motion of the paint can, the covariance matrix needs to be diagonalized, so that all redundancies are removed, and the largest variances of the set of measurements are ordered from largest to smallest. In other words, the system has reduced to its principal components. The SVD computes this for us by working in the appropriate bases $U$ and $V$, where $U$ and $V$ are unitary square matrices. For this assignment, the matrix $U$ is $n \times m$, and the matrix $V$ is $m \times m$. Note that in the SVD, $X = U\Sigma V^*$, where $\Sigma$ represents a scaling matrix and corresponds to $S$ in the implementation of the PCA algorithm.

We define

$$Y = U^* X \tag{3}$$

to set up the computation of the variance in $Y$, which results in the following:

$$C_Y = \frac{1}{n-1} \Sigma^2 \tag{4}$$

where each diagonal entry in $\Sigma$ is the square root of the eigenvalues of $Y$.

The principal components projection is given by Equation 3 above.

The full details involving the derivation of these equations may be found in our textbook, but only the above results were used in completing this assignment. Please note that $S$ and $\Sigma$ are identical, and $S$ was used in the MATLAB code.

Finally, the proportional energy values of each mode were also computed to determine what percentage each mode contributed to the overall motion of the paint can.

# 3 Algorithm Implementation and Development

The algorithm used to complete this assignment is outlined below. More specific details may be seen in the MATLAB code included in Appendix B.

1. Load the test data into the work environment.
2. Determine the number of frames in each video.
3. Crop the number of frames in each video so that each video begins where the paint can attains its first maximum height along its trajectory (synchronize the starting points as closely as possible by eye).
4. Convert each frame to a grayscale image.
5. Loop through all frames in each video, and find the maximum pixel intensity in each frame.
6. Store the location of the maximum pixel intensity as $x$ and $y$ coordinates.
7. Truncate each coordinate vector so that all coordinate vectors are the same length.
8. Combine the coordinate vectors into a matrix $X$.
9. Subtract the mean value from each row of $X$.
10. Use $X$ as input to MATLAB's SVD function to produce principal components.
11. Compute energy proportion for each PCA mode.
12. Produce plots of eigenvalues and PCA modes.

# 4 Computational Results

The following figures illustrate the results of applying the algorithm outlined above. The results are displayed in the same manner for each case, in which the top two plots depict the eigenvalues of our data as linear and logarithmic plots, the middle plot depicts the first three of six modes obtained from the PCA and the columns of the matrix $V$, and the bottom plot displays the trajectory of the paint can obtained from the first three modes of the PCA from the columns of the matrix $U$, scaled by the corresponding variances contained in the matrix $S$.

For case 1, the primary direction of motion for the paint can was vertical, so it was expected that the first mode from the PCA would dominate. This can clearly be seen in Figure 1 below.

The noise introduced in the form of camera shake in case 2 very clearly influenced the results, as can be seen in Figure 2 below. Note that mode 1 still dominates, but contains more jaggedness in the trajectory plot.

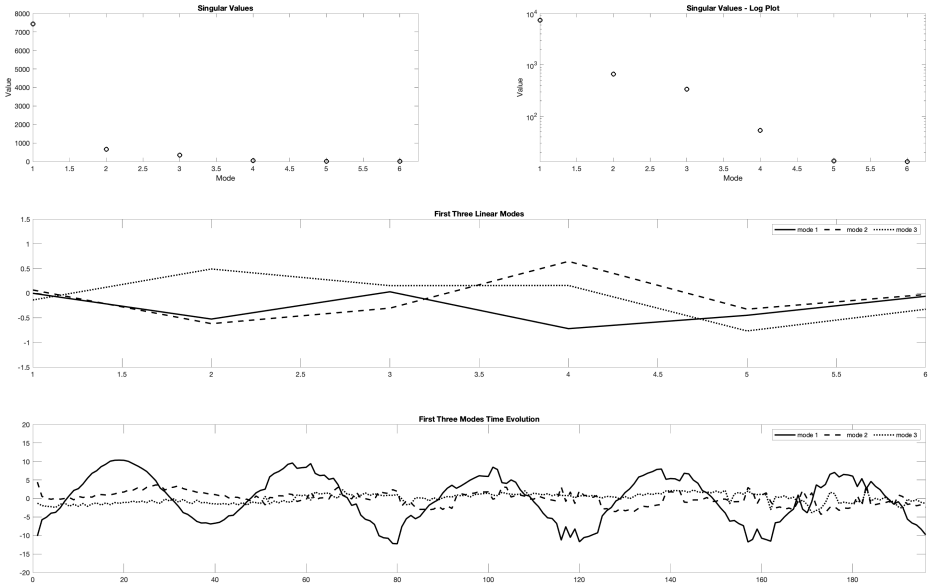


Figure 1: The above figure displays the results of applying SVD and PCA to the data collected for Case 1. The top two plots display the eigenvalues of X, the middle plot displays the first three PCA modes, and the bottom plot displays the paint can's trajectory obtained from the matrix $U$, scaled by the corresponding variance, for the first three modes. Note that the primary mode is mode 1, which displays the up and down motion of the paint can.

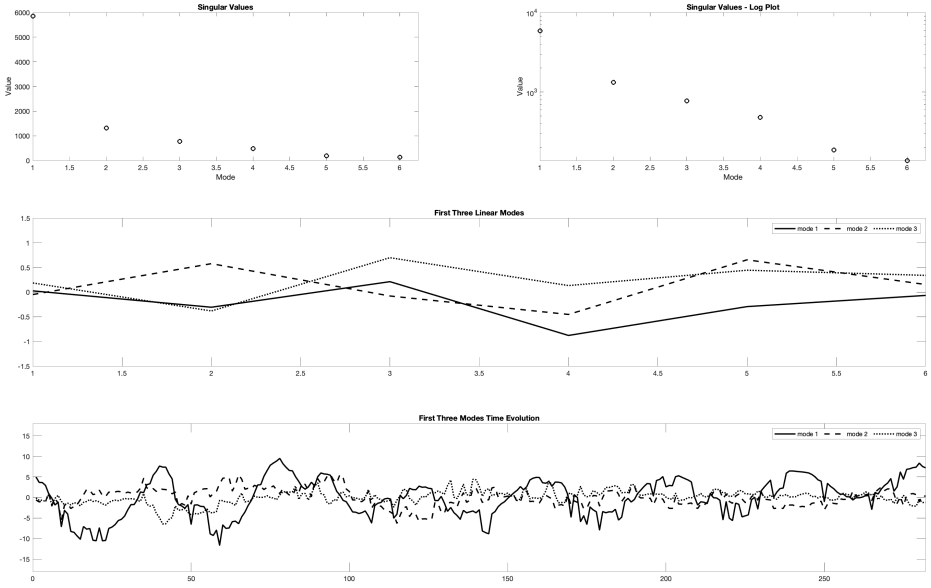


Figure 2: The above figure displays the SVD and PCA results for Case 2. Note that mode 1 still dominates, although the noise introduced some jaggedness to the plot of the trajectory.

The horizontal displacement introduced for case 3 resulted in the second mode becoming more visible in the plot of the paint can's trajectory, as expected.

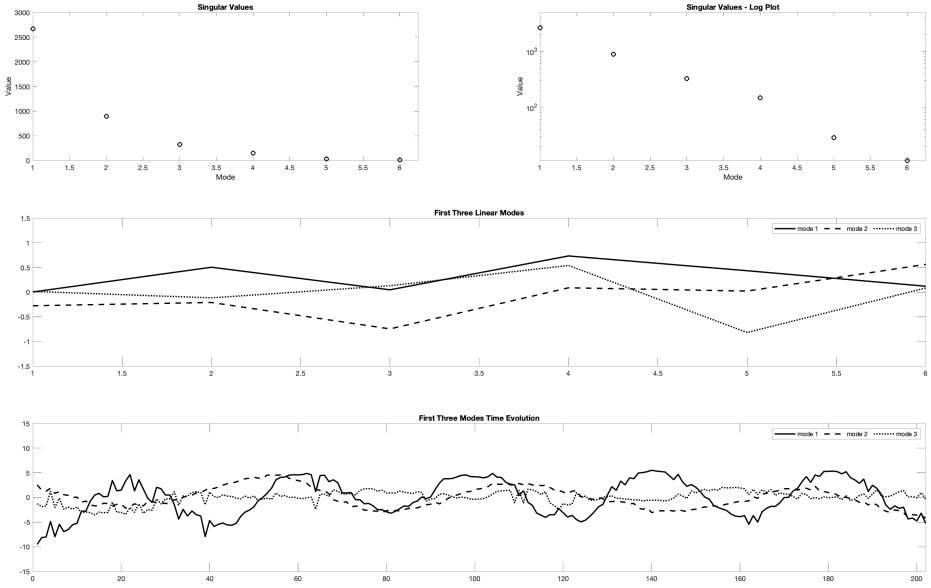


Figure 3: The above figure displays the SVD and PCA results for Case 3. Note that an additional mode appears more prominently in the plot of the paint can's trajectory.

The rotational component introduced for case 4 did not seem to have a large effect on the dominant modes of the paint can's trajectory, which was unexpected. The predominant motion of the paint can remained vertical, with small horizontal and rotational components present.

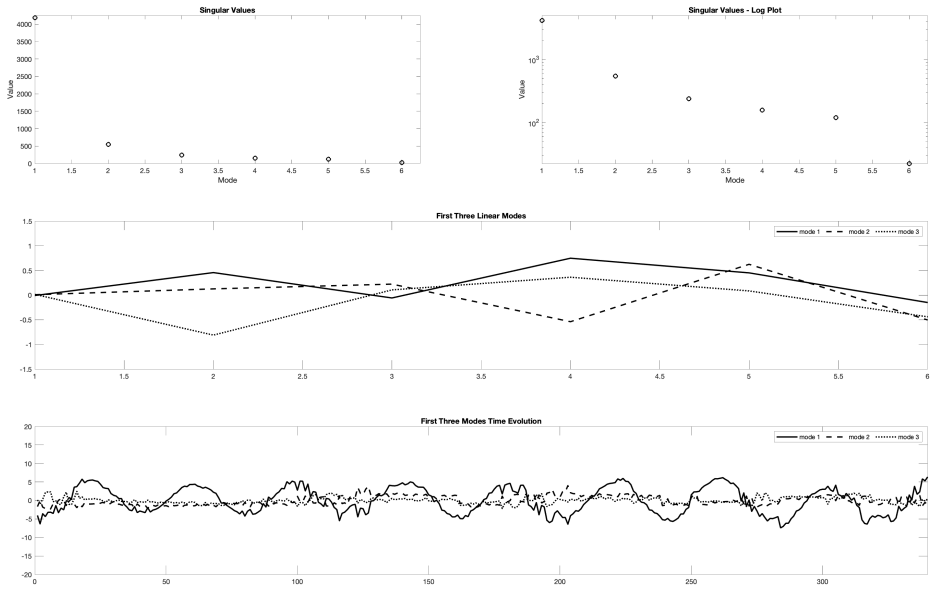


Figure 4: The above figure displays the SVD and PCA results for Case 4. Note that only the first mode dominates in the plot of the trajectory, even though a rotational component was added to the paint can's motion for this case.

The proportional energy values for each mode were also computed, and in all cases, the first mode comprised at least forty percent of the overall motion, while the first three modes comprised at least seventy-five percent of the overall motion.

# 5 Summary and Conclusions

The application of SVD and PCA enabled me to extract the overall motion of the paint can from redundant or noisy data sets. Tracking the brightest or maximum intensity pixel in each video frame for each camera seems to have been effective, since the motion of the paint can may be easily seen in the figures displayed on the preceding pages.

# References

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

# 6 Appendix A: MATLAB Functions

The following MATLAB functions were utilized in the course of completing this assignment. More detailed descriptions of each function may be found in the MATLAB documentation. Note that some functions do not appear in the code included in Appendix B; these functions were used only to view the videos prior to analyzing and cropping them.

- `implay(video)` allows playback of a video file. This was used to play the video recording of the paint can for each camera.
- `img = rgb2gray(data)` converted each frame of a video to a grayscale image from a truecolor image to allow for easier processing.
- `double(img)` converts image data from type uint8 to type double for performing mathematical operations.
- `repmat(mn,1,n)` creates a $1 \times n$ matrix containing values $mn$. This was used to subtract mean values from each row of a matrix containing coordinate data for the paint can.
- `[U,S,V] = svd(X)` performs the SVD on the matrix $X$. This was used to compute the unitary matrices $U$ and $V$, along with the diagonalized covariance matrix $S$.
- `diag(S)` creates a vector containing the diagonal entries of the matrix $S$. This was used to extract variances for each mode in the PCA.

# 7 Appendix B: MATLAB Code

The following code was used to complete this assignment. Lines of code whose function or purpose may not be obvious have comments adjacent to them containing brief descriptions or explanations.

Please visit `https://github.com/jpratt84/amath582` to view or download the MATLAB code below.

```
%AMATH 582 HW 3
clear all; close all; clc;

%% Test 1 Ideal Case

%Load video frames into tensors rows x columms x depth x frame
%depth of 3 -> rgb values
load cam1_1;
X11 = vidFrames1_1;
load cam2_1;
X21 = vidFrames2_1;
load cam3_1;
X31 = vidFrames3_1;

%Determine number of frames for each camera data set
f1 = size(X11,4);
f2 = size(X21,4);
f3 = size(X31,4);

%initialize vectors to store coordinates of max pixel values (flashlight
%on top of paint can)
x11 = [];
y11 = []; %first max at frame 31

%Create empty array to store each frame as grayscale image
cam11 = zeros(size(X11,1),size(X11,2),size(X11,4));

%Loop through each video for each camera, extract max pixel values in x and
%y
for k=1:f1
    img = rgb2gray(X11(:,:,:,k));
    cam11(:,:,k) = double(img); %Convert img from uint8 to double for math
    %Crop each frame to narrow pixel range for paint can motion
    img(:,1:300) = 0;
    img(:,400:end) = 0;
    img(1:200,:) = 0;
```

```
    [mx11,idx11] = max(img(:)); %find max pixel values
    [row11,col11] = ind2sub(size(img),idx11); %convert linear indices to
    %subscripts
    x11 = [x11 col11]; %store x,y coordinates of each max pixel value in
    %each frame
    y11 = [y11 row11];
end

x21 = [];
y21 = []; %first max at frame 43
cam21 = zeros(size(X21,1),size(X21,2),size(X21,4));

for k=1:f2
    img = rgb2gray(X21(:,:,:,k));
    cam21(:,:,k) = double(img);
    img(:,1:250) = 0;
    img(:,350:end) = 0;
    img(1:100,:) = 0;
    img(375:end,:) = 0;
    [mx21,idx21] = max(img(:));
    [row21,col21] = ind2sub(size(img),idx21);
    x21 = [x21 col21];
    y21 = [y21 row21];
end

x31 = [];
y31 = []; %first max at frame 30
cam31 = zeros(size(X31,1),size(X31,2),size(X31,4));

for k=1:f3
    img = rgb2gray(X31(:,:,:,k));
    cam31(:,:,k) = double(img);
    img(:,1:250) = 0;
    img(:,500:end) = 0;
    img(1:200,:) = 0;
    img(350:end,:) = 0;
    [mx31,idx31] = max(img(:));
    [row31,col31] = ind2sub(size(img),idx31);
    x31 = [x31 col31];
    y31 = [y31 row31];
end

%Each video contains different number of frames. Initialize each
%coordinate vector at first max value in x or y; crop coordinate
%vectors to match length of shortest vector

x11 = x11(31:end);
y11 = y11(31:end);
x21 = x21(43:end);
y21 = y21(43:end);
x31 = x31(30:end);
y31 = y31(30:end);

fmin = min([length(x11) length(x21) length(x31)]);
x11 = x11(1:fmin);
y11 = y11(1:fmin);
x21 = x21(1:fmin);
y21 = y21(1:fmin);
x31 = x31(1:fmin);
y31 = y31(1:fmin);

%Combine coordinate vectors into matrix for PCA
X1 = [x11; y11; x21; y21; x31; y31];

%Subtract mean from each row
```

```
[m,n] = size(X1); %compute data size
mn = mean(X1,2); %compute mean for each row
X1 = X1-repmat(mn,1,n); %subtract mean

%Use SVD Method to produce principal components
[u1,s1,v1] = svd(X1'/sqrt(n-1),'econ'); %econ option removes rows, cols
%containing only zeros
lambda1 = diag(s1).^2; %produce diagonal variances
Y1 = u1*X1; %produce the principal components projection

%Compute energy proportions for all modes
sig1 = diag(s1);
energy1 = zeros(1,6);
energy1(1) = sig1(1)/sum(sig1);
for jj=2:6
    energy1(jj) = sum(sig1(1:jj))/sum(sig1(:));
end

%Plot PCA projection
%Each column of v1 is projection of data onto each pca mode
%Each column of u1 is paint can's displacement along each mode
figure(1)
xs = 1:6;
ts = 1:fmin;

subplot(3,2,1), plot(lambda1,'ko','Linewidth',[1.5]),
axis([1 6.25 0 8e3]); xlabel('Mode'), ylabel('Value');
title('Singular Values');
subplot(3,2,2), semilogy(lambda1,'ko','Linewidth',[1.5]),
axis([1 6.25 0 1e4]); xlabel('Mode'), ylabel('Value');
title('Singular Values - Log Plot');
subplot(3,1,2), plot(xs,v1(:,1),'k',xs,v1(:,2),'k--',xs,v1(:,3),'k:',...
    'Linewidth',[2]);
ylim([-1.5 1.5]),
legend('mode 1','mode 2','mode 3','Orientation','horizontal');
title('First Three Linear Modes');
subplot(3,1,3), plot(ts,sig1(1).*u1(:,1),'k',ts,sig1(2).*u1(:,2),'k--',...
    ts,sig1(3).*u1(:,3),'k:','Linewidth',[2]);
xlim([0 fmin]), ylim([-20 20]);
legend('mode 1','mode 2','mode 3','Orientation','horizontal');
title('First Three Modes Time Evolution'); %each mode scaled by
%corresponding singular value

%% Test 2 Noise Added

%load video frames into tensors rows x columms x page x frame
%depth of 3 -> rgb values
load cam1_2;
X12 = vidFrames1_2;
load cam2_2;
X22 = vidFrames2_2;
load cam3_2;
X32 = vidFrames3_2;

f12 = size(X12,4); %number of frames
f22 = size(X22,4);
f32 = size(X32,4);

x12 = [];
y12 = []; %first max frame 33
cam12 = zeros(size(X12,1),size(X12,2),size(X12,4));

for k=1:f12
    img = rgb2gray(X12(:,:,:,k));
    cam12(:,:,k) = double(img);
```

```
    img(:,1:300) = 0;
    img(:,400:end) = 0;
    img(1:200,:) = 0;
    [mx12,idx12] = max(img(:));
    [row12,col12] = ind2sub(size(img),idx12);
    x12 = [x12 col12];
    y12 = [y12 row12];
end

x22 = [];
y22 = []; %first max frame 20
cam22 = zeros(size(X22,1),size(X22,2),size(X22,4));

for k=1:f22
    img = rgb2gray(X22(:,:,:,k));
    cam22(:,:,k) = double(img);
    img(:,1:200) = 0;
    img(:,400:end) = 0;
    img(1:70,:) = 0;
    img(370:end,:) = 0;
    [mx22,idx22] = max(img(:));
    [row22,col22] = ind2sub(size(img),idx22);
    x22 = [x22 col22];
    y22 = [y22 row22];
end

x32 = [];
y32 = []; %first max frame 38
cam32 = zeros(size(X32,1),size(X32,2),size(X32,4));

for k=1:f32
    img = rgb2gray(X32(:,:,:,k));
    cam32(:,:,k) = double(img);
    img(:,1:250) = 0;
    img(:,500:end) = 0;
    img(1:200,:) = 0;
    img(350:end,:) = 0;
    [mx32,idx32] = max(img(:));
    [row32,col32] = ind2sub(size(img),idx32);
    x32 = [x32 col32];
    y32 = [y32 row32];
end

%Each video contains different number of frames. Initialize each
%coordinate vector at first max value in x or y; crop coordinate
%vectors to match length of shortest vector

x12 = x12(33:end);
y12 = y12(33:end);
x22 = x22(20:end);
y22 = y22(20:end);
x32 = x32(38:end);
y32 = y32(38:end);

fmin2 = min([length(x12) length(x22) length(x32)]);
x12 = x12(1:fmin2);
y12 = y12(1:fmin2);
x22 = x22(1:fmin2);
y22 = y22(1:fmin2);
x32 = x32(1:fmin2);
y32 = y32(1:fmin2);

X2 = [x12; y12; x22; y22; x32; y32];

%Subtract mean from each row
```

```
[m,n] = size(X2); %compute data size
mn = mean(X2,2); %compute mean for each row
X2 = X2 - repmat(mn,1,n); %subtract mean

%Use SVD Method to produce principal components
[u2,s2,v2] = svd(X2'/sqrt(n-1),'econ'); %econ option removes rows, cols
%containing only zeros
lambda2 = diag(s2).^2; %produce diagonal variances
Y2 = u2*X2; %produce the principal components projection

%Compute energy proportions for all modes
sig2 = diag(s2);
energy2 = zeros(1,6);
energy2(1) = sig2(1)/sum(sig2);
for jj=2:6
    energy2(jj) = sum(sig2(1:jj))/sum(sig2(:));
end

%Plot PCA projection
%Each column of v2 is projection of data onto each pca mode
%Each column of u2 is paint can's displacement along each mode
figure(2)
xs = 1:6;
ts = 1:fmin2;

subplot(3,2,1), plot(lambda2,'ko','Linewidth',[1.5]),
axis([1 6.25 0 6e3]); xlabel('Mode'), ylabel('Value');
title('Singular Values');
subplot(3,2,2), semilogy(lambda2,'ko','Linewidth',[1.5]),
axis([1 6.25 0 1e4]); xlabel('Mode'), ylabel('Value');
title('Singular Values - Log Plot');
subplot(3,1,2), plot(xs,v2(:,1),'k',xs,v2(:,2),'k--',xs,v2(:,3),'k:',...
    'Linewidth',[2]);
ylim([-1.5 1.5]),
legend('mode 1','mode 2','mode 3','Orientation','horizontal');
title('First Three Linear Modes');
subplot(3,1,3), plot(ts,sig2(1).*u2(:,1),'k',ts,sig2(2).*u2(:,2),'k--',...
    ts,sig2(3).*u2(:,3),'k:','Linewidth',[2]);
xlim([0 fmin2]), ylim([-18 18]);
legend('mode 1','mode 2','mode 3','Orientation','horizontal');
title('First Three Modes Time Evolution'); %each mode scaled by
%corresponding singular value

%% Test 3 Horizontal Displacement

%load video frames into tensors rows x columms x page x frame
%depth of 3 -> rgb values
load cam1_3;
X13 = vidFrames1_3;
load cam2_3;
X23 = vidFrames2_3;
load cam3_3;
X33 = vidFrames3_3;

f13 = size(X13,4); %number of frames
f23 = size(X23,4);
f33 = size(X33,4);

x13 = [];
y13 = []; %first max frame 38
cam13= zeros(size(X13,1),size(X13,2),size(X13,4));

for k=1:f13
    img = rgb2gray(X13(:,:,:,k));
    cam13(:,:,k) = double(img);
```

```
    img(:,1:300) = 0;
    img(:,400:end) = 0;
    img(1:200,:) = 0;
    [mx13,idx13] = max(img(:));
    [row13,col13] = ind2sub(size(img),idx13);
    x13 = [x13 col13];
    y13 = [y13 row13];
end

x23 = [];
y23 = []; %second max frame 66
cam23 = zeros(size(X23,1),size(X23,2),size(X23,4));

for k=1:f23
    img = rgb2gray(X23(:,:,:,k));
    cam23(:,:,k) = double(img);
    img(:,1:250) = 0;
    img(:,350:end) = 0;
    img(1:90,:) = 0;
    img(360:end,:) = 0;
    [mx23,idx23] = max(img(:));
    [row23,col23] = ind2sub(size(img),idx23);
    x23 = [x23 col23];
    y23 = [y23 row23];
end

x33 = [];
y33 = []; %first max frame 32
cam33 = zeros(size(X33,1),size(X33,2),size(X33,4));

for k=1:f33
    img = rgb2gray(X33(:,:,:,k));
    cam33(:,:,k) = double(img);
    img(:,1:250) = 0;
    img(:,500:end) = 0;
    img(1:200,:) = 0;
    img(350:end,:) = 0;
    [mx33,idx33] = max(img(:));
    [row33,col33] = ind2sub(size(img),idx33);
    x33 = [x33 col33];
    y33 = [y33 row33];
end

%Each video contains different number of frames. Initialize each
%coordinate vector at first max value in x or y; crop coordinate
%vectors to match length of shortest vector

x13 = x13(38:end);
y13 = y13(38:end);
x23 = x23(66:end);
y23 = y23(66:end);
x33 = x33(32:end);
y33 = y33(32:end);

fmin3 = min([length(x13) length(x23) length(x33)]);
x13 = x13(1:fmin3);
y13 = y13(1:fmin3);
x23 = x23(1:fmin3);
y23 = y23(1:fmin3);
x33 = x33(1:fmin3);
y33 = y33(1:fmin3);

X3 = [x13; y13; x23; y23; x33; y33];

%Subtract mean from each row
```

```
[m,n] = size(X3); %compute data size
mn = mean(X3,2); %compute mean for each row
X3 = X3 - repmat(mn,1,n); %subtract mean

%Use SVD Method to produce principal components
[u3,s3,v3] = svd(X3'/sqrt(n-1),'econ'); %econ option removes rows, cols
%containing only zeros
lambda3 = diag(s3).^2; %produce diagonal variances
Y3 = u3*X3; %produce the principal components projection

%Compute energy proportions for all modes
sig3 = diag(s3);
energy3 = zeros(1,6);
energy3(1) = sig3(1)/sum(sig3);
for jj=2:6
    energy3(jj) = sum(sig3(1:jj))/sum(sig3(:));
end

%Plot PCA projection
%Each column of v2 is projection of data onto each pca mode
%Each column of u2 is paint can's displacement along each mode
figure(3)
xs = 1:6;
ts = 1:fmin3;

subplot(3,2,1), plot(lambda3,'ko','Linewidth',[1.5]),
axis([1 6.25 0 3e3]); xlabel('Mode'), ylabel('Value');
title('Singular Values');
subplot(3,2,2), semilogy(lambda3,'ko','Linewidth',[1.5]),
axis([1 6.25 0 5e3]); xlabel('Mode'), ylabel('Value');
title('Singular Values - Log Plot');
subplot(3,1,2), plot(xs,v3(:,1),'k',xs,v3(:,2),'k--',xs,v3(:,3),'k:',...
    'Linewidth',[2]);
ylim([-1.5 1.5]),
legend('mode 1','mode 2','mode 3','Orientation','horizontal');
title('First Three Linear Modes');
subplot(3,1,3), plot(ts,sig3(1).*u3(:,1),'k',ts,sig3(2).*u3(:,2),'k--',...
    ts,sig3(3).*u3(:,3),'k:','Linewidth',[2]);
xlim([0 fmin3]), ylim([-15 15]);
legend('mode 1','mode 2','mode 3','Orientation','horizontal');
title('First Three Modes Time Evolution'); %each mode scaled by
%corresponding singular value

%% Test 4 Horizontal Displacement and Rotation

%load video frames into tensors rows x columms x page x frame
%depth of 3 -> rgb values
load cam1_4;
X14 = vidFrames1_4;
load cam2_4;
X24 = vidFrames2_4;
load cam3_4;
X34 = vidFrames3_4;

f14 = size(X14,4); %number of frames
f24 = size(X24,4);
f34 = size(X34,4);

x14 = [];
y14 = []; %first max frame 53
cam14= zeros(size(X14,1),size(X14,2),size(X14,4));

for k=1:f14
    img = rgb2gray(X14(:,:,:,k));
    cam14(:,:,k) = double(img);
```

```
    img(:,1:300) = 0;
    img(:,450:end) = 0;
    img(1:175,:) = 0;
    img(425:end,:) = 0;
    [mx14,idx14] = max(img(:));
    [row14,col14] = ind2sub(size(img),idx14);
    x14 = [x14 col14];
    y14 = [y14 row14];
end

x24 = [];
y24 = []; %first max frame 60
cam24 = zeros(size(X24,1),size(X24,2),size(X24,4));

for k=1:f24
    img = rgb2gray(X24(:,:,:,k));
    cam24(:,:,k) = double(img);
    img(:,1:225) = 0;
    img(:,425:end) = 0;
    img(1:90,:) = 0;
    img(360:end,:) = 0;
    [mx24,idx24] = max(img(:));
    [row24,col24] = ind2sub(size(img),idx24);
    x24 = [x24 col24];
    y24 = [y24 row24];
end

x34 = [];
y34 = []; %first max frame 52
cam34 = zeros(size(X34,1),size(X34,2),size(X34,4));

for k=1:f34
    img = rgb2gray(X34(:,:,:,k));
    cam34(:,:,k) = double(img);
    img(:,1:300) = 0;
    img(:,475:end) = 0;
    img(1:150,:) = 0;
    img(275:end,:) = 0;
    [mx34,idx34] = max(img(:));
    [row34,col34] = ind2sub(size(img),idx34);
    x34 = [x34 col34];
    y34 = [y34 row34];
end

%Each video contains different number of frames. Initialize each
%coordinate vector at first max value in x or y; crop coordinate
%vectors to match length of shortest vector

x14 = x14(53:end);
y14 = y14(53:end);
x24 = x24(60:end);
y24 = y24(60:end);
x34 = x34(52:end);
y34 = y34(52:end);

fmin4 = min([length(x14) length(x24) length(x34)]);
x14 = x14(1:fmin4);
y14 = y14(1:fmin4);
x24 = x24(1:fmin4);
y24 = y24(1:fmin4);
x34 = x34(1:fmin4);
y34 = y34(1:fmin4);

X4 = [x14; y14; x24; y24; x34; y34];
```

```
%Subtract mean from each row
[m,n] = size(X4); %compute data size
mn = mean(X4,2); %compute mean for each row
X4 = X4 - repmat(mn,1,n); %subtract mean

%Use SVD Method to produce principal components
[u4,s4,v4] = svd(X4'/sqrt(n-1),'econ'); %econ option removes rows, cols
%containing only zeros
lambda4 = diag(s4).^2; %produce diagonal variances
Y4 = u4*X4; %produce the principal components projection

%Compute energy proportions for all modes
sig4 = diag(s4);
energy4 = zeros(1,6);
energy4(1) = sig4(1)/sum(sig4);
for jj=2:6
    energy4(jj) = sum(sig4(1:jj))/sum(sig4(:));
end

%Plot PCA projection
%Each column of v4 is projection of data onto each pca mode
%Each column of u4 is paint can's displacement along each mode
figure(4)
xs = 1:6;
ts = 1:fmin4;

subplot(3,2,1), plot(lambda4,'ko','Linewidth',[1.5]),
axis([1 6.25 0 4.25e3]); xlabel('Mode'), ylabel('Value');
title('Singular Values');
subplot(3,2,2), semilogy(lambda4,'ko','Linewidth',[1.5]),
axis([1 6.25 0 5e3]); xlabel('Mode'), ylabel('Value');
title('Singular Values - Log Plot');
subplot(3,1,2), plot(xs,v4(:,1),'k',xs,v4(:,2),'k--',xs,v4(:,3),'k:',...
    'Linewidth',[2]);
ylim([-1.5 1.5]),
legend('mode 1','mode 2','mode 3','Orientation','horizontal');
title('First Three Linear Modes');
subplot(3,1,3), plot(ts,sig4(1).*u4(:,1),'k',ts,sig4(2).*u4(:,2),'k--',...
    ts,sig4(3).*u4(:,3),'k:','Linewidth',[2]);
xlim([0 fmin4]), ylim([-20 20]);
legend('mode 1','mode 2','mode 3','Orientation','horizontal');
title('First Three Modes Time Evolution'); %each mode scaled by
%corresponding singular value
```