AMATH 582 Homework 3

Christopher Liu

February 21, 2020

Abstract

Principal component analysis (PCA) is a powerful tool in allowing us to identify dynamics of interest in a system. We have been presented with recordings of a paint can moving on a spring from 3 different cameras for 4 different test scenarios. PCA will be employed to empirically extract the governing equations of motion of the paint can by studying the linear proper orthogonal modes of the system. By analyzing the footage, we will highlight the strengths of PCA in identifying key dynamics as well as demonstrate how PCA can fail when the data has high levels of noise.

1 Introduction and Overview

For this assignment, we have been tasked with analyzing the movement of a paint can on an spring recorded on 3 different cameras from 3 distinct angles. Utilizing the prinicipal component analysis (PCA), we will then be able to empirically extract the governing equations of motion of the paint can.

The behavior of the paint can is altered in 4 different tests that were conducted. In doing so, we hope to highlight the strengths and limitations of PCA. Test 1 is the ideal case where the paint can is oscillating up and down in 1 dimension. Test 2 is similar to the ideal case but instead has noise generated by shaking the camera while it is recording. Test 3 includes the addition of horizontal movement to the paint can. Finally, Test 4 has both vertical and horizontal displacement and rotation.

2 Theoretical Background

2.1 Singular Value Decomposition

Every matrix $A \in^{m\times n}$ has a singular value decomposition (SVD) given by Equation (1) [1].

$$A = U\Sigma V^* \tag{1}$$

where $U \in^{m\times m}$ is unitary, $V \in^{n\times n}$ is unitary and $\Sigma \in^{m\times n}$ is diagonal. Furthermore, σ_j, the components of Σ are uniquely determined for all A. The SVD allows us to describe multiplication of a matrix by a vector as a unitary transformation (V^*) followed by a stretch (Σ) and another unitary transformation (U)

2.2 Calculating the SVD

The SVD is easy to compute if we consider the following matrices AA^T and A^TA,

$$A^TA = (U\Sigma V^*)^T(U\Sigma V^*) = V\Sigma^2V^* \tag{2}$$

$$AA^T = (U\Sigma V^*)(U\Sigma V^*)^T = U\Sigma^2U^* \tag{3}$$

Multiplying (2) and (3) on the right by V and U respectively, we are presented with 2 eigenvalue problems

$$A^TAV = V\Sigma^2 \tag{4}$$

$$AA^TU = U\Sigma^2 \tag{5}$$

So if we find the normalized eigenvectors of AA^T and A^TA, we are able to construct U and V and the square-root of the eigenvalues will be the singular values.

2.3 Relationship between SVD and Variance

The covariance matrix of A, C_A, is a square symmetric $m \times m$ where $A \in^{m\times n}$ contains the measured data of the system given by 6 with n measurements and m measuring positions. The diagonal terms of C_A are the variances for particular measurements. We are interested when the variances are large as they are assumed to correspond to the dynamics of interest. The off-diagonal elements are the covariances between measurements. A large covariance represents a high level of redundancy between the two measurements and vice versa. By using the covariance matrix, we can identify the dynamics of interest corresponding to maximal variance and remove redundancy [1].

$$C_A = \frac{1}{n-1}AA^T \tag{6}$$

We can relate the SVD and the covariance matrix in the following way,

$$C_Y = \frac{1}{n-1}AA^T = \frac{1}{n-1}\Sigma^2 \tag{7}$$

where $A = U\Sigma V^*$ and $Y = U^*A$. So by calculating the SVD of A we can determine a relative relationship between the variances of each transformed measurement (ie. which ones are much larger than the others). Furthermore, the columns of U contain the linear proper orthogonal modes (POD) which constitute the orthonormal expansion basis of interest and the columns of V show their evolution in time.

3 Algorithm Implementation and Development

There are 2 main steps that are taken to carry out the analysis. The first is to track the movement of the paint can with each frame in 2 dimensions for each camera and test. The second is to combine the vectors of the paint can's position with each frame into a matrix and to compute its SVD.

A pink flashlight is placed on top of the can and allows us to track the movement in each frame. Depending on the recording and test, we can track either the light emitted by the flashlight (in color or black and white) or its pink body. In order to use our algorithm for tracking the paint can, we will need a few initial parameters. Namely, an initial guess, window width, length of recording in frames, and what color/object to track. The initial guess and the choice of object to track is determined by watching the original footage and the window width is guessed and refined with trial and error. After finding the initial parameters, we can use our algorithm to track the can which is described in detail in Algorithm 1.

Algorithm 1: Tracking the Paint Can

```
Set center of search window to initial guess
Set max intensity of pixel to 0
for i = 2 : (length of recording in frames) do
    Extract ith frame of recording
    for x = (x center - width) : (x center - width) do
        for y = (y center - width) : (y center - width) do
            if intensity of (x, y) >(max intensity) then
                Set max intensity to intensity at (x, y)
            end if
        end for
    end for
    Set center of search window to (x, y) of max intensity
    Store location of max intensity
end for
```

3 different functions were defined to track the paint can. They follow the same logic as 1 but look at different intensity values corresponding to white, when the recording is in color or black and white, and pink. After we measure the positions, we plot their positions to center the peaks and troughs of the oscillation and truncate them so that they are of the same length. We construct the measurement matrix $A \in \mathbb{R}^{6\times n}$ where n is the number of frames and each row corresponds to the X or Y coordinate of the 3 cameras. We will also subtract the mean off each vector. Finally, we take the SVD of A^T as it is more conventional to solve an over-determined system. Note this flips the interpretation of U and V

4 Computational Results

4.1 Test 1: Ideal Case

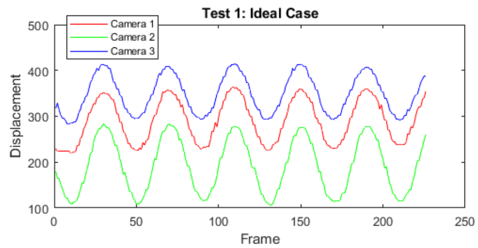

Figure 1: Tracked positions for test 1

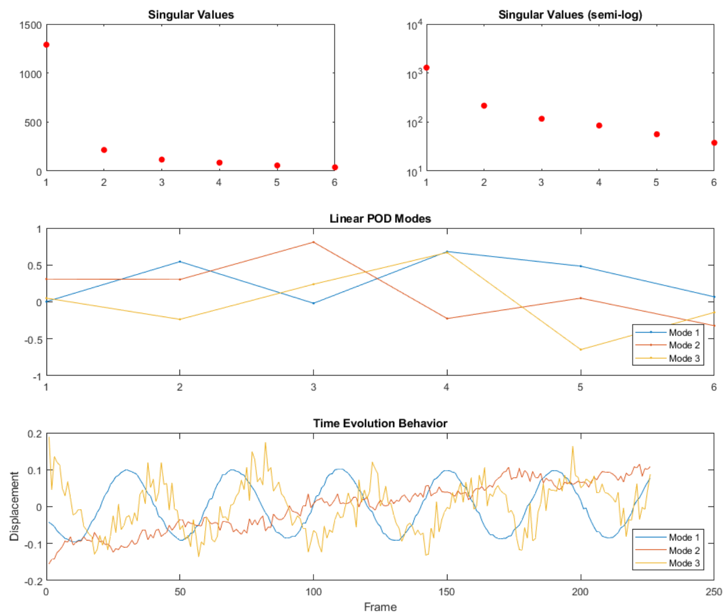

Figure 2: The singular values, linear PODs and the time evolution for Test 1

For the ideal case, we find that our tracking algorithm does a fairly good job. There is 1 large singular value which is what we expect since the paint can is moving in 1-D. If we look at Mode 1 in the Linear POD Mode plot, we see that each camera roughly only has one non-zero component which corresponds to movement in 1-D. Finally, looking at the time evolution behavior, we see that mode 1 exhibits simple harmonic motion which is again what we would expect for Test 1.

4.2 Test 2: Noisy Case

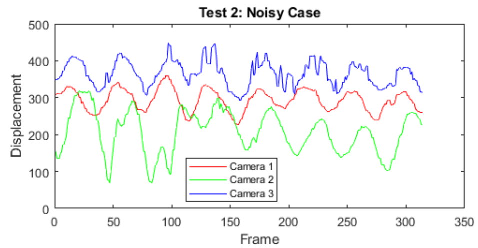

Figure 3: Tracked positions for test 2

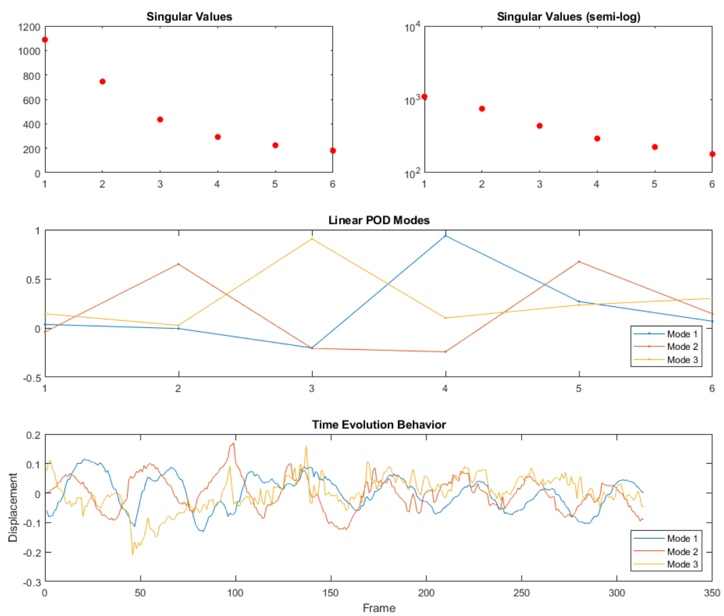

Figure 4: The singular values, linear PODs and the time evolution for Test 2

The tracking algorithm does not perform as well for Test 2 but this is expected as this is the noisy case. Nonetheless, simple harmonic motion can still be made out. The addition of noise has increased the values of the last 5 singular values but there is still 1 large value. For the linear modes and the time evolution behavior, it is no longer apparent that the paint can is only moving in 1-D. The noise obscures the dynamics of the system.

4.3 Test 3: Horizontal Displacement

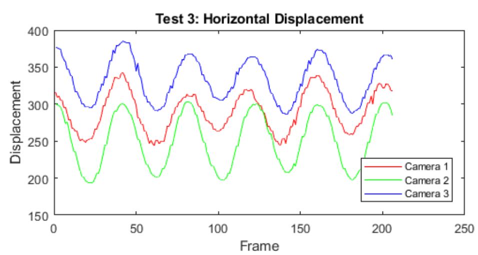

Figure 5: Tracked positions for test 3

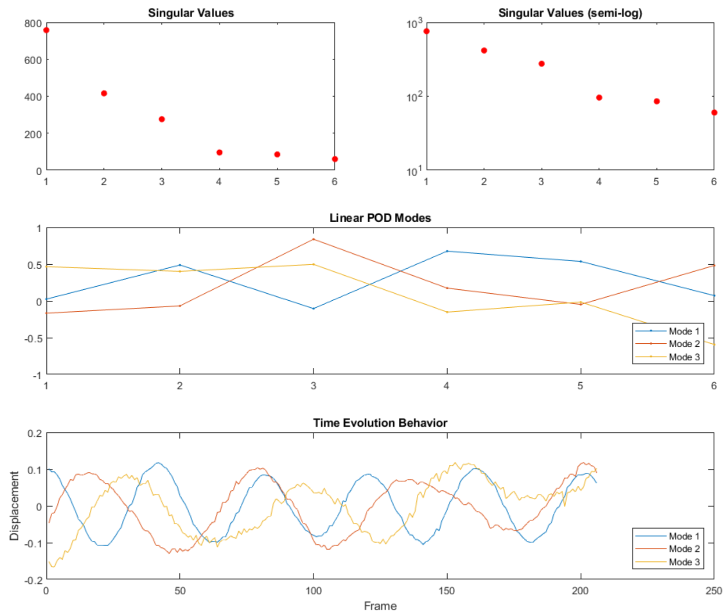

Figure 6: The singular values, linear PODs and the time evolution for Test 3

Our tracking algorithm seems to do well even with the addition of the horizontal displacement. When inspecting the original footage, the addition of horizontal displacement has added displacement in X and Y directions. The camera angle and an imperfect initial push gives us movement in 3-D. This is reflected in the singular values where the largest one corresponds to the harmonic motion in the Z-direction and the next 2 correspond to X and Y directions.The singular values tell us that the simple harmonic motion in Z is still the dominant behavior. Looking at the time evolution of the modes, we can now observe oscillatory behavior in modes 2 and 3 in addition to the harmonic motion in mode 1.

4.4 Test 4: Horizontal Displacement and Rotation

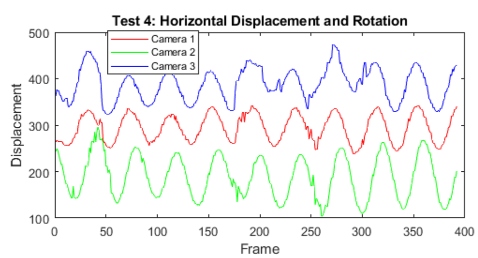

Figure 7: Tracked positions for test 4

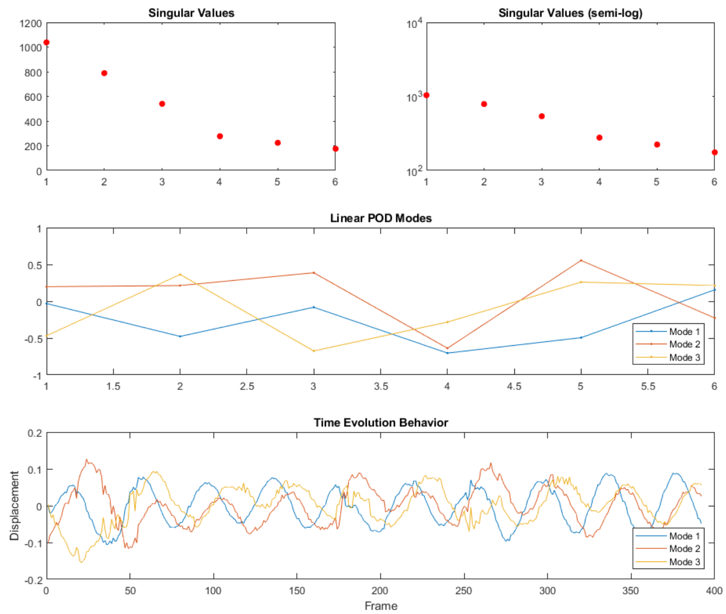

Figure 8: The singular values,linear PODs and the time evolution for Test 4

The tracking algorithm struggles a little more with Test 4 but still produces decent results. Upon inspection of the footage, we observe that the paint can itself is rotating which makes tracking the flashlight more difficult. When the flashligh was not visible, the tracker would instead track the bottom of the paint can. Again, as expected we have 3 large singular values corresponding to displacement in 3-D. The harmonic oscillation in the Z-direction is still the dominant behavior of the system. The time evolution also reflects oscillation in 3-D however it is a little 'noisy' due to difficulties in tracking the flashlight.

5 Summary and Conclusions

Through the use of PCA, we have been able to empirically extract the governing equations of motion of the paint can. We were able to confirm our findings by watching the original footage and as such have been able to highlight the strength of PCA to extract meaningful characteristics from data. From analyzing a noisy data set, we were also able to show the limitations of PCA when working with imperfect data.

References

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

Appendix A MATLAB Functions

Below are the key MATLAB functions implemented.

- `imshow(vidFrames)` displays the picture given by the RGB values in a figure
- `[U,S,V]=svd(A,0)` returns the reduced singular value decomposition of A
- `rgb2gray(vidFrames)` converts an RGB image to a grayscale image.
- `diag(S)` returns the diagonal elements of S in a vector

Appendix B MATLAB Code

Github Repo: https://github.com/chrismhl/AMATH582/tree/master/Homework3

```
clear all; clc ;close all;

load('cam1_1.mat')
load('cam1_2.mat')
load('cam1_3.mat')
load('cam1_4.mat')
load('cam2_1.mat')
load('cam2_2.mat')
load('cam2_3.mat')
load('cam2_4.mat')
load('cam3_1.mat')
load('cam3_2.mat')
load('cam3_3.mat')
load('cam3_4.mat')

%%
% Tracking for camera 1 part 1
[x1_1, y1_1] = track_paintcan_gray(vidFrames1_1,[321,228],[20,20,20,20]);

%%
% Playing Camera 1 Part 1
for i=1:226
    figure(1)
    imshow(vidFrames1_1(:,:,:,i))
    hold on
    plot(x1_1(i),y1_1(i),'r*')
    hold off
end

%%
% Tracking for camera 2 part 1
[x2_1, y2_1] = track_paintcan_gray(vidFrames2_1,[274,274],[20,20,20,20]);

%%
% Play Camera 2 Part 1
for i=1:284
    figure(2)
    imshow(uint8(vidFrames2_1(:,:,:,i)))
    hold on
    plot(x2_1(i),y2_1(i),'r*')
    hold off
end

%%
% Tracking for camera 3 part 1
[x3_1, y3_1] = track_paintcan_gray(vidFrames3_1,[318,271],[15,15,15,15]);

%%
% Play Camera 3 Part 1
for i=1:232
    figure(2)
    imshow(uint8(vidFrames3_1(:,:,:,i)))
    hold on
```

```
    plot(x3_1(i),y3_1(i),'r*')
    hold off
end

%%
%Plot position of all 3 cameras, test 1
close all

figure(3)

subplot(3,1,1)
plot(1:226,y1_1,'r')
hold on
plot(1:284,y2_1,'g')
plot(1:232,x3_1,'b')
hold off
legend('Camera 1', 'Camera 2', 'Camera 3')

%truncated
subplot(3,1,2)
plot(1:226,y1_1,'r')
hold on
plot(1:226,y2_1(10:235),'g')
plot(1:226,x3_1(1:226),'b')
hold off
legend('Camera 1', 'Camera 2', 'Camera 3')
title('Truncated')
%%
% SVD part 1
A1(1,:) = x1_1 - mean(x1_1);
A1(2,:) = y1_1 - mean(y1_1);
A1(3,:) = x2_1(10:235) - mean(x2_1(10:235));
A1(4,:) = y2_1(10:235) - mean(y2_1(10:235));
A1(5,:) = x3_1(1:226) - mean(x3_1(1:226));
A1(6,:) = y3_1(1:226) - mean(y3_1(1:226));
[U1,s1,V1] = svd(A1.',0);

sig1 = diag(s1);
energy1_1 = sig1(1)/sum(sig1);
energy2_1 = sum(sig1(1:2))/sum(sig1);
energy3_1 = sum(sig1(1:3))/sum(sig1);

figure(5)
subplot(3,2,1)
plot(1:length(sig1),sig1,'r.','MarkerSize',20)
title('Singular Values')
subplot(3,2,2)
plot(1:length(sig1),sig1,'r.','MarkerSize',20)
set(gca, 'YScale', 'log')
title('Singular Values (semi-log)')

subplot(3,1,2)
plot(1:6,V1(:,1),'.-',1:6,V1(:,2),'.-',1:6,V1(:,3),'.-')
legend('Mode 1','Mode 2','Mode 3','Location','southeast')
xticks([1 2 3 4 5 6])
title('Linear POD Modes')
```

```
subplot(3,1,3)
plot(1:226,U1(:,1),1:226,U1(:,2),1:226,U1(:,3))
legend('Mode 1','Mode 2','Mode 3','Location','southeast')
ylabel('Displacement')
xlabel('Frame')
title('Time Evolution Behavior')
%%
% Tracking for camera 1 part 2
[x1_2, y1_2] = track_paintcan(vidFrames1_2,[325,308],[17,17*0.9,15,15*0.9]);

%%
% Tracking Camera 1 Part 2
for i=1:314
    figure(2)
    imshow(uint8(vidFrames1_2(:,:,:,i)))
    hold on
    plot(x1_2(i),y1_2(i),'r*')
    hold off
end

%%
% Tracking for camera 2 part 2
[x2_2, y2_2] = track_paintcan(vidFrames2_2,[314,357],[34,34,34,35]);

%%
% Tracking Camera 2 Part 2
for i=1:356
    figure(2)
    imshow(uint8(vidFrames2_2(:,:,:,i)))
    hold on
    plot(x2_2(i),y2_2(i),'r*')
    hold off
end

%%
% Tracking for camera 3 part 2
[x3_2, y3_2] = track_paintcan_gray(vidFrames3_2,[349,245],[44,45,44,44]);

%%
% Tracking Camera 3 Part 2
for i=1:327
    figure(2)
    imshow(uint8(vidFrames3_2(:,:,:,i)))
    hold on
    plot(x3_2(i),y3_2(i),'r*')
    hold off
end

%%
%Plot position of all 3 cameras part 2
close all

figure(5)

subplot(2,1,1)
```

```
plot(1:314,y1_2,'r')
hold on
plot(1:356,y2_2,'g')
plot(1:327,x3_2,'b')
hold off
legend('Camera 1', 'Camera 2', 'Camera 3')

%truncated
subplot(2,1,2)
plot(1:314,y1_2,'r')
hold on
plot(1:314  ,y2_2(15:328),'g')
plot(1:314,x3_2(1:314),'b')
hold off
legend('Camera 1', 'Camera 2', 'Camera 3')
title('Truncated')

%%
% SVD part 2
A2 = zeros(6,314);
A2(1,:) = x1_2 - mean(x1_2);
A2(2,:) = y1_2 - mean(y1_2);
A2(3,:) = x2_2(15:328) - mean(x2_2(15:328));
A2(4,:) = y2_2(15:328) - mean(y2_2(15:328));
A2(5,:) = x3_2(1:314) - mean(x3_2(1:314));
A2(6,:) = y3_2(1:314) - mean(y3_2(1:314));
[U2,s2,V2] = svd(A2.',0);

sig2 = diag(s2);
energy1_2 = sig2(1)/sum(sig2);
energy2_2 = sum(sig2(1:2))/sum(sig2);
energy3_2 = sum(sig2(1:3))/sum(sig2);

figure(6)
subplot(3,2,1)
plot(1:length(sig2),sig2,'r.','MarkerSize',20)
title('Singular Values')
subplot(3,2,2)
plot(1:length(sig2),sig2,'r.','MarkerSize',20)
set(gca, 'YScale', 'log')
title('Singular Values (semi-log)')

subplot(3,1,2)
plot(1:6,V2(:,1),'.-',1:6,V2(:,2),'.-',1:6,V2(:,3),'.-')
legend('Mode 1','Mode 2','Mode 3','Location','southeast')
xticks([1 2 3 4 5 6])
title('Linear POD Modes')

subplot(3,1,3)
plot(1:314,U2(:,1),1:314,U2(:,2),1:314,U2(:,3))
legend('Mode 1','Mode 2','Mode 3','Location','southeast')
ylabel('Displacement')
xlabel('Frame')
title('Time Evolution Behavior')

%%
```

```
% Tracking for camera 1 part 3, tracking pink

[x1_3, y1_3] = track_paintcan_pink(vidFrames1_3,[330,290],[20,20,20,20]);
%%
% Tracking Camera 1 Part 3
for i=1:239
    figure(2)
    imshow(uint8(vidFrames1_3(:,:,:,i)))
    hold on
    plot(x1_3(i),y1_3(i),'r*')
    hold off
end

%%
% Tracking for camera 2 part 3, tracking pink

[x2_3, y2_3] = track_paintcan_pink(vidFrames2_3,[252,294],[20,20,20,20]);
%%
% Tracking Camera 2 Part 3
for i=1:281
    figure(2)
    imshow(uint8(vidFrames2_3(:,:,:,i)))
    hold on
    plot(x2_3(i),y2_3(i),'r*')
    hold off
end

%%
% Tracking for camera 3 part 3
[x3_3, y3_3] = track_paintcan_gray(vidFrames3_3,[353,229],[17,17,17,17]);

%%
% Tracking Camera 3 Part 3
for i=1:237
    figure(2)
    imshow(uint8(vidFrames3_3(:,:,:,i)))
    hold on
    plot(x3_3(i),y3_3(i),'r*')
    hold off
end

%%
%Plot position of all 3 cameras part 3
close all

figure(7)

subplot(2,1,1)
plot(1:239,y1_3,'r')
hold on
plot(1:281,y2_3,'g')
plot(1:237,x3_3,'b')
hold off
legend('Camera 1', 'Camera 2', 'Camera 3')

%truncated
```

```
subplot(2,1,2)
plot(1:206,y1_3(20:225),'r')
hold on
plot(1:206   ,y2_3(5:210),'g')
plot(1:206,x3_3(10:215),'b')
hold off
legend('Camera 1', 'Camera 2', 'Camera 3')
title('Truncated')

%%
% SVD part 3
A3 = zeros(6,206);
A3(1,:) = x1_3(20:225) - mean(x1_3(20:225));
A3(2,:) = y1_3(20:225) - mean(y1_3(20:225));
A3(3,:) = x2_3(5:210) - mean(x2_3(5:210));
A3(4,:) = y2_3(5:210) - mean(y2_3(5:210));
A3(5,:) = x3_3(10:215) - mean(x3_3(10:215));
A3(6,:) = y3_3(10:215) - mean(y3_3(10:215));
[U3,s3,V3] = svd(A3.',0);
sig3 = diag(s3);

energy1_3 = sig3(1)/sum(sig3);
energy2_3 = sum(sig3(1:2))/sum(sig3);
energy3_3 = sum(sig3(1:3))/sum(sig3);

figure(8)
subplot(3,2,1)
plot(1:length(sig3),sig3,'r.','MarkerSize',20)
title('Singular Values')
subplot(3,2,2)
plot(1:length(sig3),sig3,'r.','MarkerSize',20)
set(gca, 'YScale', 'log')
title('Singular Values (semi-log)')

subplot(3,1,2)
plot(1:6,V3(:,1),'.-',1:6,V3(:,2),'.-',1:6,V3(:,3),'.-')
legend('Mode 1','Mode 2','Mode 3','Location','southeast')
xticks([1 2 3 4 5 6])
title('Linear POD Modes')

subplot(3,1,3)
plot(1:206,U3(:,1),1:206,U3(:,2),1:206,U3(:,3))
legend('Mode 1','Mode 2','Mode 3','Location','southeast')
ylabel('Displacement')
xlabel('Frame')
title('Time Evolution Behavior')


%%
% Tracking for camera 1 part 4
[x1_4, y1_4] = track_paintcan_pink(vidFrames1_4,[402,263],[44,45,44,44]);

%%
% Tracking Camera 1 Part 4
for i=1:392
    figure(2)
```

```
    imshow(uint8(vidFrames1_4(:,:,:,i)))
    hold on
    plot(x1_4(i),y1_4(i),'r*')
    hold off
end

%%
% Tracking for camera 2 part 4
[x2_4, y2_4] = track_paintcan_pink(vidFrames2_4,[244,243],[44,45,44,44]);

%%
% Tracking Camera 2 Part 4
for i=1:405
    figure(2)
    imshow(uint8(vidFrames2_4(:,:,:,i)))
    hold on
    plot(x2_4(i),y2_4(i),'r*')
    hold off
end

%%
% Tracking for camera 3 part 4
[x3_4, y3_4] = track_paintcan(vidFrames3_4,[363,244],[40,40,30,30]);

%%
% Tracking Camera 3 Part 4
for i=1:394
    figure(2)
    imshow(uint8(vidFrames3_4(:,:,:,i)))
    hold on
    plot(x3_4(i),y3_4(i),'r*')
    hold off
end

%%
%Plot position of all 3 cameras part 4
close all

figure(9)

subplot(2,1,1)
plot(1:392,y1_4,'r')
hold on
plot(1:405,y2_4,'g')
plot(1:394,x3_4,'b')
hold off
legend('Camera 1', 'Camera 2', 'Camera 3')

%truncated
subplot(2,1,2)
plot(1:392,y1_4,'r')
hold on
plot(1:392   ,y2_4(1:392),'g')
plot(1:392,x3_4(1:392),'b')
hold off
legend('Camera 1', 'Camera 2', 'Camera 3')
```

```
title('Truncated')

%%
% SVD part 4
A4 = zeros(6,392);
A4(1,:) = x1_4 - mean(x1_4);
A4(2,:) = y1_4 - mean(y1_4);
A4(3,:) = x2_4(1:392) - mean(x2_4(1:392));
A4(4,:) = y2_4(1:392) - mean(y2_4(1:392));
A4(5,:) = x3_4(1:392) - mean(x3_4(1:392));
A4(6,:) = y3_4(1:392) - mean(y3_4(1:392));
[U4,s4,V4] = svd(A4.',0);

sig4 = diag(s4);
energy1_4 = sig4(1)/sum(sig4);
energy2_4 = sum(sig4(1:2))/sum(sig4);
energy3_4 = sum(sig4(1:3))/sum(sig4);

figure(10)
subplot(3,2,1)
plot(1:length(sig4),sig4,'r.','MarkerSize',20)
title('Singular Values')
subplot(3,2,2)
plot(1:length(sig4),sig4,'r.','MarkerSize',20)
title('Singular Values (semi-log)')
set(gca, 'YScale', 'log')

subplot(3,1,2)
plot(1:6,V4(:,1),'.-',1:6,V4(:,2),'.-',1:6,V4(:,3),'.-')
legend('Mode 1','Mode 2','Mode 3','Location','southeast')
title('Linear POD Modes')

subplot(3,1,3)
plot(1:392,U4(:,1),1:392,U4(:,2),1:392,U4(:,3))
legend('Mode 1','Mode 2','Mode 3','Location','southeast')
ylabel('Displacement')
xlabel('Frame')
title('Time Evolution Behavior')

%%
% Plot all position vectors

figure(11)
plot(1:226,y1_1,'r')
hold on
plot(1:226,y2_1(10:235),'g')
plot(1:226,x3_1(1:226),'b')
hold off
title('Test 1: Ideal Case')
lgd = legend('Camera 1', 'Camera 2', 'Camera 3','Location','southeast');
lgd.FontSize =7;
ylabel('Displacement')
xlabel('Frame')

figure(12)
plot(1:314,y1_2,'r')
```

```
hold on
plot(1:314   ,y2_2(15:328),'g')
plot(1:314,x3_2(1:314),'b')
lgd = legend('Camera 1', 'Camera 2', 'Camera 3','Location','southeast');
lgd.FontSize =7;
hold off
title('Test 2: Noisy Case')
ylabel('Displacement')
xlabel('Frame')


figure(13)
plot(1:206,y1_3(20:225),'r')
hold on
plot(1:206   ,y2_3(5:210),'g')
plot(1:206,x3_3(10:215),'b')
hold off
title('Test 3: Horizontal Displacement')
ylabel('Displacement')
xlabel('Frame')
lgd = legend('Camera 1', 'Camera 2', 'Camera 3','Location','southeast');
lgd.FontSize =7;

figure(14)
plot(1:392,y1_4,'r')
hold on
plot(1:392   ,y2_4(1:392),'g')
plot(1:392,x3_4(1:392),'b')
hold off
title('Test 4: Horizontal Displacement and Rotation')
ylabel('Displacement')
xlabel('Frame')
lgd = legend('Camera 1', 'Camera 2', 'Camera 3','Location','southeast');
lgd.FontSize =7;
```

B.1 Additional Functions

```
%vidFrames is the .dat file corresponding to the recording
%guess is the initial guess of the flashlight in [x y] from frame 1
%width is the search width of the tracking algorithm
%returns vectors xp and yp which are the positions of the paintcan with
%   each frame
function [xp,yp] = track_paintcan(vidFrames,guess,width)
dim = size(vidFrames);
L = dim(4);     %length of video in frames
xp = zeros(L,1);
yp = zeros(L,1);
maxloc = [0,0];
maxval = 0;

init_guess = guess;
xp(1) = init_guess(1);
yp(1) = init_guess(2);

%center the first search window to the initial point
center = init_guess;
```

```
for f = 2:L
    frame = double(vidFrames(:,:,:,f));
     for x = center(1)-width(1):center(1)+width(2)
        for y = center(2)-width(3):center(2)+width(4)
            point = frame(y,x,:);
            colorsum = point(1,1,1)+point(1,1,2)+point(1,1,3);
            if colorsum > maxval
                maxloc = [x,y];
                maxval = colorsum;
            end
        end
    end
    xp(f) = maxloc(1);
    yp(f) = maxloc(2);
    maxloc = [0,0];
    maxval = 0;

    center(1) = xp(f);
    center(2) = yp(f);
end
```

```
%vidFrames is the .dat file corresponding to the recording
%guess is the initial guess of the flashlight in [x y] from frame 1
%width is the search width of the tracking algorithm
%returns vectors xp and yp which are the positions of the paintcan with
%   each frame
function [xp,yp] = track_paintcan_gray(vidFrames,guess,width)
dim = size(vidFrames);
L = dim(4); %length of video in frames
xp = zeros(L,1);
yp = zeros(L,1);
maxloc = [0,0];
maxval = 0;

init_guess = guess;
xp(1) = init_guess(1);
yp(1) = init_guess(2);

%center the first search window to the initial point
center = init_guess;

for f = 2:L
    frame = double(rgb2gray(vidFrames(:,:,:,f)));
    for x = center(1)-width(1):center(1)+width(2)
        for y = center(2)-width(3):center(2)+width(4)
            point = frame(y,x,:);
            if point > maxval
                maxloc = [x,y];
                maxval = point;
            end
        end
    end
    xp(f) = maxloc(1);
    yp(f) = maxloc(2);
    maxloc = [0,0];
    maxval = 0;
```

```

    center(1) = xp(f);
    center(2) = yp(f);
end
```

```
%vidFrames is the .dat file corresponding to the recording
%guess is the initial guess of the flashlight in [x y] from frame 1
%width is the search width of the tracking algorithm
%returns vectors xp and yp which are the positions of the paintcan with
%   each frame
function [xp,yp] = track_paintcan_pink(vidFrames,guess,width)
dim = size(vidFrames);
L = dim(4); %length of video in frames
xp = zeros(L,1);
yp = zeros(L,1);
maxloc = [0,0];
maxred = 0;

init_guess = guess;
xp(1) = init_guess(1);
yp(1) = init_guess(2);

%center the first search window to the initial point
center = init_guess;

for f = 2:L
    frame = double(vidFrames(:,:,:,f));
     for x = center(1)-width(1):center(1)+width(2)
        for y = center(2)-width(3):center(2)+width(4)
            point = frame(y,x,:);
            if ((point(1,1,1) > maxred)&(point(1,1,2)<150)&(point(1,1,3)<150))
                maxloc = [x,y];
                maxred = point;
            end
        end
    end
    xp(f) = maxloc(1);
    yp(f) = maxloc(2);
    maxloc = [0,0];
    maxred = 0;

    center(1) = xp(f);
    center(2) = yp(f);
end
```