Principal Component Analysis

Nick Moore

February 6, 2017

Abstract

Principal component analysis is a commonly applied method for data analysis. It identifies variables known as principal components that account for variance in the data set. The number of principal components is less than or equal to the number of original variables. Herein PCA is applied to data extracted from videos of a paint can attached to a spring oscillating up and down. The resultant principle components are used to determine if the motion of paint can is captured by the principal components, and if so how many it takes to capture most of the variance in the data set.

Sec. I. Introduction and Overview

The goal of this report is to explore principal component analysis (PCA) and its results with a real data set. Videos of a paint can attached to a string oscillating up and down were analyzed and had PCA applied to it. Four different cases were analyzed. The cases are labeled as N_M where M is 1 through 4. N_1 is the ideal case with motion only in one direction, N_2 has camera shake to add noise, N_3 introduces motion in the x direction, and N_4 has x motion and rotational motion. Three videos were analyzed from each case, taken from three different angles. The position of the paint can was tracked with the trackPosition function. The position vectors were then formatted into a new matrix A for each case. PCA is then applied to the matrix A after being diagonalized with singular value decomposition (SVD). Matlab 2015b was used for all of the work presented. The functions used and the entire code can be found in Appendices A & B respectively.

Sec. II. Theoretical Background

Principal component analysis (PCA) is a statistical method that creates linear combinations of the original variables called principal components. The principal components form a new basis set and each one captures a portion of the variance from the original data set. The principal components of a matrix A are the eigenvectors of AA^T.[1] The first principal component captures the most variance, which decreases as principal component number increases. PCA can be applied through one of two methods of diagonalization, which is necessary because it reduces the dimensionality of the data. Eigenvectors and eigenvalues or singular value decomposition (SVD) can be used for diagonalization. For the work presented here SVD was used. Briefly, SVD is a matrix decomposition method that results in three new, simpler matrices U, Σ, and V. For PCA a fourth matrix, Y, is generated. It is the principal component basis defined below, where A is the data matrix.[1]

$$Y = U^* A$$

The matrix Y contains all of the principal component information, but the variance in Y (C_Y) needs to be calculated to determine how much variance is captured by each principal component. This is shown in the equations below.[1]

$$C_Y = \frac{1}{n-1} YY^T$$

$$C_Y = \frac{1}{n-1}(U^*X)(U^*X)^T$$
$$C_Y = \frac{1}{n-1}U^*(XX^T)U$$
$$C_Y = \frac{1}{n-1}U^*U\Sigma^2UU^*$$
$$C_Y = \frac{1}{n-1}\Sigma^2$$

Because we are concerned with the variance in Y being captured by the Σ matrix, A needs to be modified. A is a matrix with m by n size and it is divided by the square root of n -1 to before undergoing SVD, shown below. The Σ matrix is squared to create a matrix that contains the variances of the principal components along its diagonal.

$$[U, \Sigma, V] = SVD\left(\frac{A}{\sqrt{n-1}}\right)$$

Sec. III. Algorithm Implementation and Development

In order to analyze the motion of the paint can with PCA the position of the paint can had to be extracted from the videos and reorganized in a new matrix A. X_1 and Y_1 contain the position data in the x and y directions for camera 1.

$$A = \begin{bmatrix} X_1 \\ Y_1 \\ X_2 \\ Y_2 \\ X_3 \\ Y_3 \end{bmatrix}$$

Since the paint can and the flashlight attached to it are both white they could be tracked by noting the position of the white pixels in each frame of the video files. The function trackPosition was used to do this (lines 168-182). The function accepted two inputs the video file and the minimum pixel intensity percentage that is tracked (line 168). The function then calculates the minimum grayscale pixel intensity based on the maximum pixel intensity and the percentage input (line 173). The video files and frames have different max pixel intensities so this allows for the function track a specified upper end. For all of the analysis presented the pixels within 5% of maximum intensity were tracked. The position of the white pixels (max intensity on grayscale) were tracked using the find function and assigned to the vectors x_max and y_max (line 174). The position data from x_max and y_max was clustered using k-means function with k = 1 and then assigned to the variables x and y, giving the average position of the pixels (line 176). The final tracking step was to mean center the x and y data, which was performed simply by subtracting the mean of x and y from themselves (lines 180-181). The function then output the vectors x and y giving the mean centered position of the paint can in both x and y directions.

Once the positon data had been extracted from the videos it had to be placed in a new matrix for PCA. This resulted in some unintended difficulties due to the length of the videos from different angles and other white objects in the frames. To work around these problems the size and length of the video frames were cropped (example on line 13). The video files are four dimensional matrices with the dimensions containing information on the height, width, RGB, and frame number, respectively. The height and width of each video was cropped to remove as much of the frame as possible with the entire paint present in every frame. The frames of the video used were also cropped for two reasons. The

video files have to be the same length in order to place them all in the new matrix, and the start points were chosen to align the phase of the paint can's oscillation. This was done to minimize the variance between the datasets of the different camera angles as much as possible. For every case the third video was taken at a 90° angle compared to the first two. To correct for this the x and y variables were switched so that the rotation of the camera angle could be accounted for (line 20). The x and y positions from each camera angle were then assembled into a new matrix, A, where the rows are the x and y position vectors, as shown below (lines 82-83). The matrix A then undergoes PCA (line 85). PCA is performed through the SVD method shown in the previous section. With the PCA results for the four different cases plots of position, modes, eigenvalues, and principal components were created (lines 140-166).

Sec IV. Computational Results

With PCA of the videos the goal is to show that the harmonic oscillation of the paint is captured by the principal components. As stated previously the first step was to track the position of the paint can. The x and y position of the paint can vs. time (as frame number) is plotted below in figure 1, where the red line is the x position and the blue line is the y position.

For N_1, the ideal case, harmonic oscillation is clearly observed in y direction with little movement in the x direction. The noisy case (N_2) presented a challenge for the tracking function because of the camera shake present in the videos. Despite the camera shake 1_2 and 3_2 were able to track the motion well, with small amounts of noise in both directions. Case 2_2 contains noticeably more noise than the other two, but the wave like behavior can still be seen. For the last two cases the paint can moved in x direction as well as the y, which can be seen in the plots above. PCA was applied to the reformatted in matrix A.

Figure 1: Position of the Paint Can

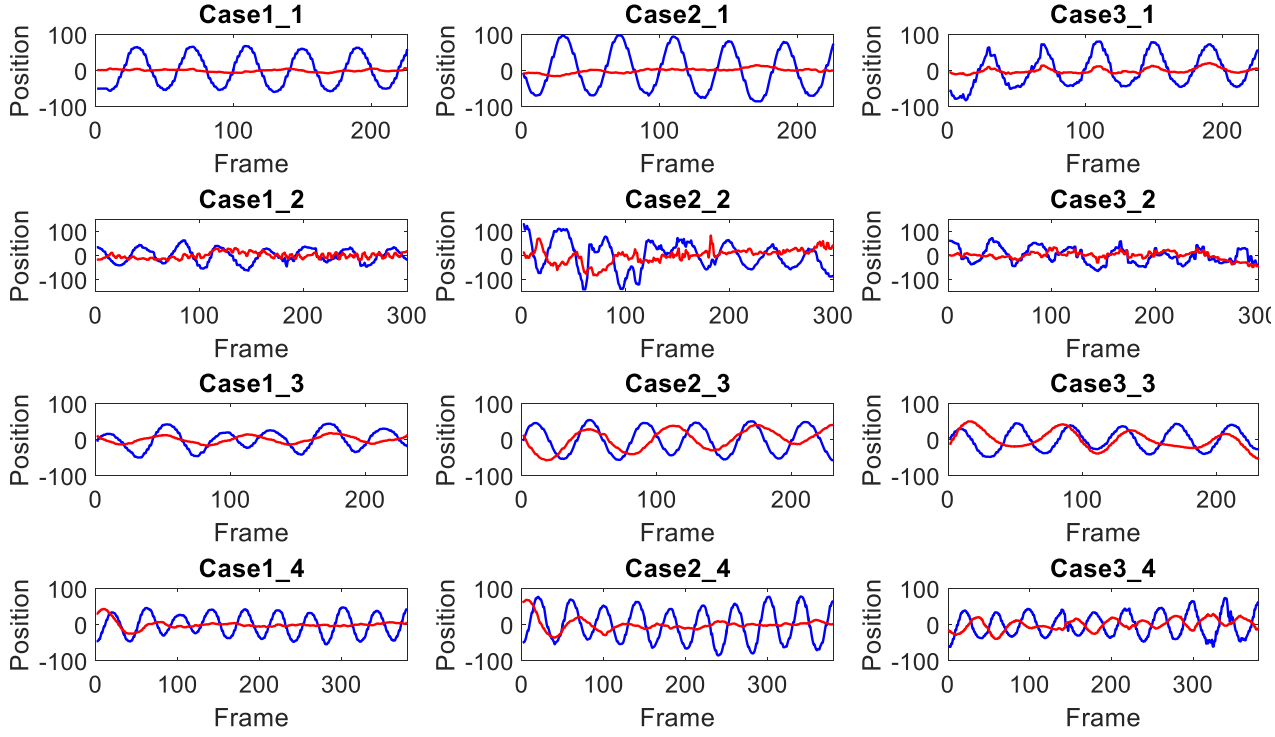

Figure 1 shows the position of the paint in y the direction (blue) and the x direction (red) for all cases.

The variance captured by the principal components is shown below in figure 2. For caseN_1 and N_4 the first principal component is able to capture a large amount of the variance, while N_2 and N_3 capture only about 60%. This is expected from the position data from figure 1 because N_1 and N_4 had little movement in the x direction, while N_2 has significant noise and N_3 shows an almost equal amount of movement in both x and y directions. The modes, shown in figure 3, also support this. The blue line is the first mode and the red line is the second mode, given by the columns of U. For N_1 the first mode shows the expected harmonic oscillation, while the second appears to be mostly noise. Both modes for N_2 contain significant noise, but the underlying oscillation can still be observed. The first two modes of N_3 both display harmonic oscillation, with one accounting for motion in the y direction and the other in the x direction. The final case, N_4, once again captures the motion in the y direction on the first mode, and the second mode accounting for the x movement seen in the first 100 frames of case1_4 and 2_4.

Since the goal of this report is to capture the oscillating movement of the paint can via PCA the principal components are integral to the interpretation of the data. Figure 4 shows the first two principal components plotted vs. variable number. The first principal component is shown in blue and the second is shown in red. The data was mean centered prior to plotting. All four of the first principal components show harmonic oscillation with cases N_1, N_3, and N_4 showing almost identical wave forms with N_2 being slightly out of phase. The position data in figure 1 shows that N_1, N_3, and N_4 all begin at a similar point in the oscillation, while N_2 is offset. This accounts for the difference observed in the first principal components. All of the second principal components show some oscillatory behavior, with N_2 and N_3 having larger fluctuations than N_1 and N_4. This is expected based on the variance captured by each principal component, shown in figure 2. For N_2 and N_3 the second principal components capture the variance due to noise and lateral motion, respectively. The last four principal components are not shown because they capture 11% or less of the variance. PCA was successful in reducing the variance of the data sets down to one or two principal components.

Figure 2: Variance Captured by Principal Components

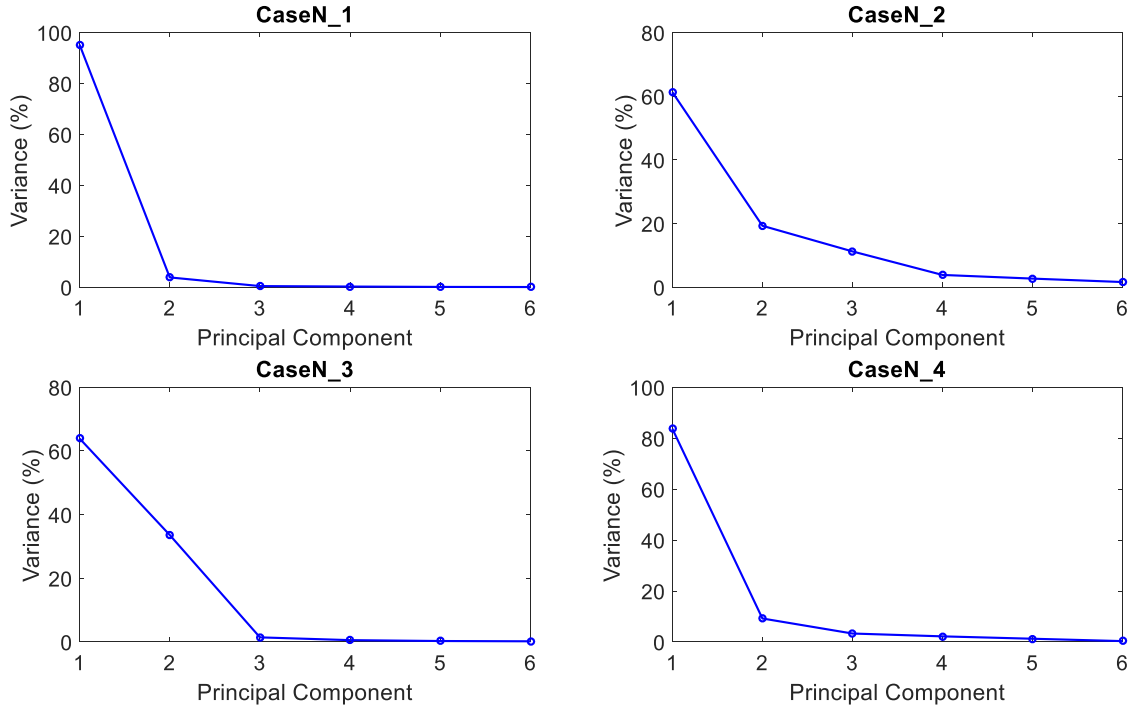

Figure 2 shows the percent variance captured of the six principal components for all cases.

Figure 3: Mode Plots

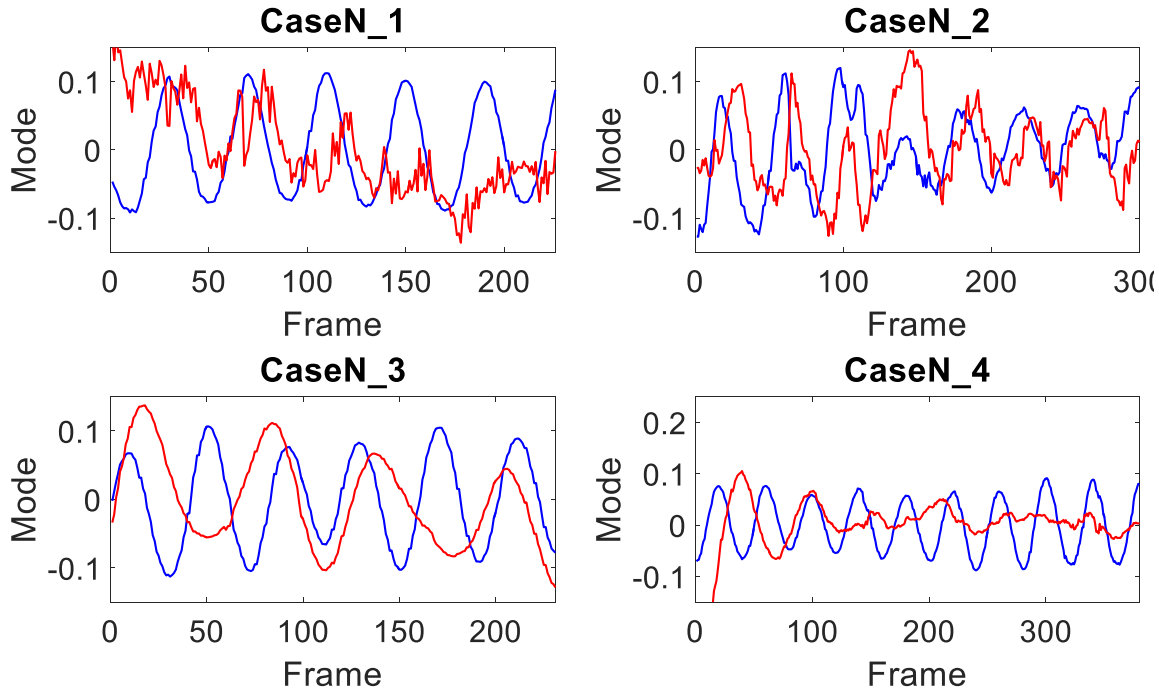

Figure 3 shows the plots of modes 1 (blue) and 2 (red) for all cases.

Figure 4: Principal Components

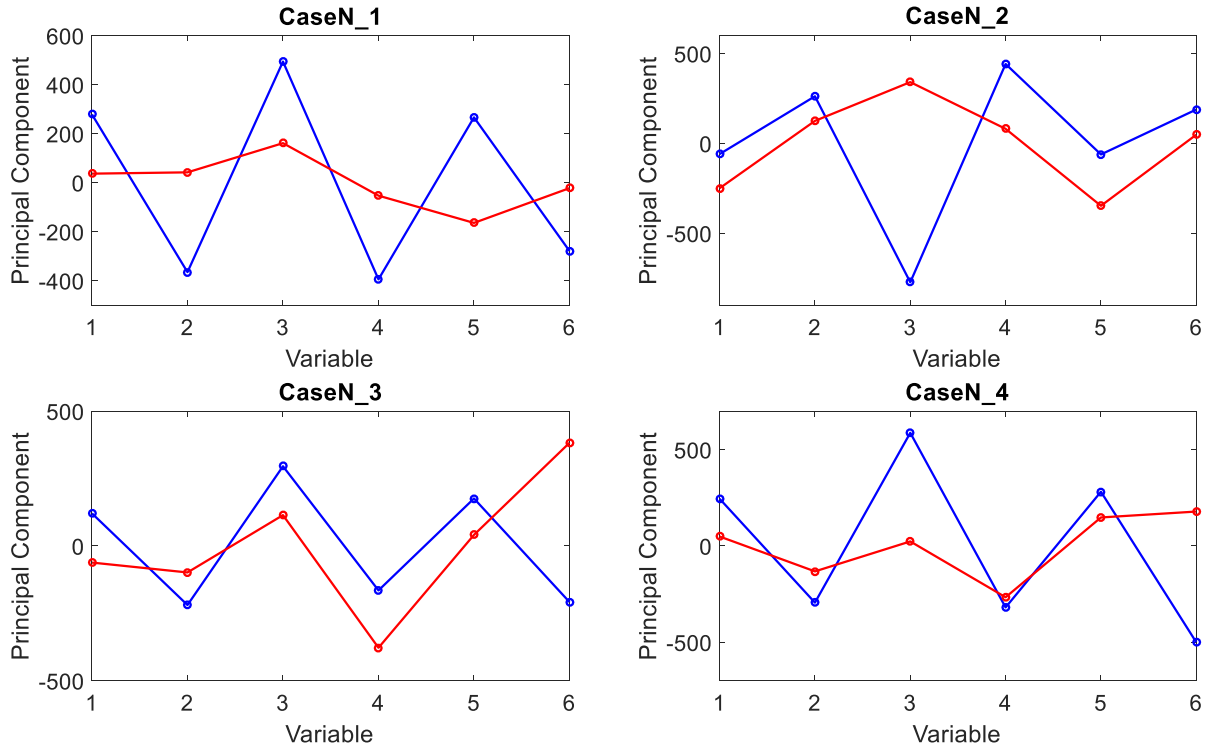

Figure 4 shows the first (blue) and second (red) principal components for all cases.

Sec. V. Summary and Conclusions

Principal component analysis was applied to videos of a paint can attached to a spring. The goal was to show that the harmonic motion of the paint could be identified and captured by PCA. Four different cases were analyzed. The first was the ideal case where the paint can moved in only one direction, the second added artificial noise via camera shake, the third also had lateral movement, and the fourth had lateral movement and rotation. The position of the paint can was tracked by recording the position of the most intense, white pixels after converting the videos to grayscale. PCA was applied to this position data after being diagonalized with singular value decomposition. The principal components in figure 4 show that the harmonic motion of the paint can was successfully captured by the first two principal components in all cases. PCA is a versatile method that is shown to be a useful method of analysis for reducing the dimensionality of a data set and capturing the variance of the data in the principal component basis set.

References

1. Kutz, J. N. *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data*; Oxford University Press, Inc.: New York, NY, USA, **2013**.

Appendix A. MATLAB Functions

[X, Y] = trackPosition(video, percent) – Tracks the position of the paint can in the video by noting the position of the 5% most intense pixels when converted to grayscale.

[idx,c] = kmeans(L, 1) – Clusters the white pixel together to give an average of the position.

Rgb2gray – Converts RGB color to grayscale where white has the highest intensity.

Appendix B. MATLAB Codes

```
%% Paint Can Tracking

% CaseN_1

% Create cells X and Y
X1 = cell(1,3);
Y1 = cell(1,3);

% Set the max frame count to the shortest video
maxCount1 = min([size(vidFrames1_1,4) size(vidFrames2_1,4) size(vidFrames3_1,4)]);

% Camera 1_1
[X1{1},Y1{1}] = trackPosition(vidFrames1_1(200:430,300:400,:,1:maxCount1), 0.95);

% Camera 2_1
[X1{2},Y1{2}] = trackPosition(vidFrames2_1(100:400,250:350,:,10:maxCount1+9), 0.95);

% Camera 3_1
```

```
% X and Y are flipped because the camera orientation is flipped
[Y1{3},X1{3}] = trackPosition(vidFrames3_1(200:350,275:500,:,1:maxCount1), 0.95);

% CaseN_2

% Create cells X and Y
X2 = cell(1,3);
Y2 = cell(1,3);

% Set the max frame count to the shortest video
maxCount2 = min([size(vidFrames1_2,4) size(vidFrames2_2,4) size(vidFrames3_2,4)]);

% Camera 1_2
[X2{1},Y2{1}] = trackPosition(vidFrames1_2(200:430,300:425,:,12:maxCount2-3), 0.95);

% Camera 2_2
[X2{2},Y2{2}] = trackPosition(vidFrames2_2(70:420,190:475,:,1:maxCount2-14), 0.95);

% Camera 3_2
% X and Y are flipped because the camera orientation is flipped
[Y2{3},X2{3}] = trackPosition(vidFrames3_2(150:350,250:500,:,15:maxCount2), 0.95);

% CaseN_3

% Create cells X and Y
X3 = cell(1,3);
Y3 = cell(1,3);

% Set the max frame count to the shortest video
maxCount3 = min([size(vidFrames1_3,4) size(vidFrames2_3,4) size(vidFrames3_3,4)]);

% Camera 1_3
[X3{1},Y3{1}] = trackPosition(vidFrames1_3(200:420,275:400,:,7:maxCount3), .95);

% Camera 2_3
[X3{2},Y3{2}] = trackPosition(vidFrames2_3(175:420,175:410,:,36:maxCount3+29), .95);

% Camera 3_3
% X and Y are flipped because the camera orientation is flipped
[Y3{3},X3{3}] = trackPosition(vidFrames3_3(160:350,250:480,:,1:maxCount3-6), .95);

% CaseN_4

% Create cells X and Y
X4 = cell(1,3);
Y4 = cell(1,3);

% Set the max frame count to the shortest video
```

```
maxCount4 = min([size(vidFrames1_4,4) size(vidFrames2_4,4) size(vidFrames3_4,4)]);

% Camera 1_4
[X4{1},Y4{1}] = trackPosition(vidFrames1_4(200:420,300:460,:,12:maxCount4), 0.95);

% Camera 2_4
[X4{2},Y4{2}] = trackPosition(vidFrames2_4(100:375,200:410,:,20:maxCount4+8), 0.95);

% Camera 3_4
% X and Y are flipped because the camera orientation is flipped
[Y4{3},X4{3}] = trackPosition(vidFrames3_4(140:300,250:500,:,14:maxCount4+2), 0.95);

%% SVD on matrix A

% CaseN_1
A1 = [X1{1}; Y1{1}; X1{2}; Y1{2}; X1{3}; Y1{3}];
A1 = transpose(A1);
[m1,n1] = size(A1);
[u1,s1,v1] = svd(A1/sqrt(n1-1),'econ');
lambda1 = diag(s1).^2;
PC_1 = u1'*A1;

% CaseN_2
A2 = [X2{1}; Y2{1}; X2{2}; Y2{2}; X2{3}; Y2{3}];
A2 = transpose(A2);
[m2,n2] = size(A2);
[u2,s2,v2] = svd(A2/sqrt(n2-1),'econ');
lambda2 = diag(s2).^2;
PC_2 = u2'*A2;

% CaseN_3
A3 = [X3{1}; Y3{1}; X3{2}; Y3{2}; X3{3}; Y3{3}];
A3 = transpose(A3);
[m3,n3] = size(A3);
[u3,s3,v3] = svd(A3/sqrt(n3-1),'econ');
lambda3 = diag(s3).^2;
PC_3 = u3'*A3;

% CaseN_4
A4 = [X4{1}; Y4{1}; X4{2}; Y4{2}; X4{3}; Y4{3}];
A4 = transpose(A4);
[m4,n4] = size(A4);
[u4,s4,v4] = svd(A4/sqrt(n4-1),'econ');
lambda4 = diag(s4).^2;
PC_4 = u4'*A4;

%% Plot of position

```

```
% CaseN_1
figure(1);
for i=1:3
    subplot(4,3,i);
    plot(1:maxCount1, X1{i}, 'b', 1:maxCount1, Y1{i}, 'r', 'Linewidth',[2]); ylabel('Position');
xlabel('Frame'); legend('X', 'Y'); title(['Case' num2str(i) '\_1']); axis([0 226 -100 100]);
end

% CaseN_2
for i=1:3
    subplot(4,3,i+3);
    plot(1:maxCount2-14, X2{i}, 'b', 1:maxCount2-14, Y2{i}, 'r', 'Linewidth',[2]); ylabel('Position');
xlabel('Frame'); legend('X', 'Y'); title(['Case' num2str(i) '\_2']); axis([0 300 -150 150]);
end

% CaseN_3
for i=1:3
    subplot(4,3,i+6);
    plot(1:maxCount3-6, X3{i}, 'b', 1:maxCount3-6, Y3{i}, 'r', 'LineWidth', [2]); ylabel('Position');
xlabel('Frame'); legend('X', 'Y'); title(['Case' num2str(i) '\_3']); axis([0 231 -100 100]);
end

% CaseN_4
for i=1:3
    subplot(4,3,i+9);
    plot(1:maxCount4-11, X4{i}, 'b', 1:maxCount4-11, Y4{i}, 'r', 'LineWidth', [2]); ylabel('Position');
xlabel('Frame'); legend('X', 'Y'); title(['Case' num2str(i) '\_4']); axis([0 380 -100 100]);
end

%% Plot of Modes

figure(2)
subplot(2,2,1); plot(1:maxCount1, u1(:,1),'b', 1:maxCount1, u1(:,2),'r', 'LineWidth', [2]);
ylabel('Mode'); xlabel('Frame'); title('CaseN\_1'); legend('Mode 1', 'Mode 2'); axis([0 226 -0.15 0.15]);
subplot(2,2,2); plot(1:maxCount2-14, u2(:,1),'b', 1:maxCount2-14, u2(:,2),'r', 'LineWidth', [2]);
ylabel('Mode'); xlabel('Frame'); title('CaseN\_2'); legend('Mode 1', 'Mode 2'); axis([0 300 -0.15 0.15]);
subplot(2,2,3); plot(1:maxCount3-6, u3(:,1), 'b', 1:maxCount3-6, u3(:,2), 'r', 'LineWidth', [2]);
ylabel('Mode'); xlabel('Frame'); title('CaseN\_3'); legend('Mode 1', 'Mode 2'); axis([0 231 -0.15 0.15]);
subplot(2,2,4); plot(1:maxCount4-11, u4(:,1), 'b', 1:maxCount4-11, u4(:,2), 'r', 'LineWidth', [2]);
ylabel('Mode'); xlabel('Frame'); title('CaseN\_4'); legend('Mode 1', 'Mode 2'); axis([0 380 -0.15 0.25]);

%% Plot of Eigenvalues (s^2)

figure(3);
subplot(2,2,1); plot(lambda1/sum(lambda1)*100,'bo-','LineWidth',[2]); ylabel('Variance Captured
(%)'); xlabel('Eigenvalue'); title('CaseN\_1');
subplot(2,2,2); plot(lambda2/sum(lambda2)*100,'bo-','LineWidth',[2]); ylabel('Variance Captured
(%)'); xlabel('Eigenvalue'); title('CaseN\_2');
```

```
153. subplot(2,2,3); plot(lambda3/sum(lambda3)*100,'bo-','LineWidth',[2]); ylabel('Variance Captured
(%)'); xlabel('Eigenvalue'); title('CaseN\_3');
154. subplot(2,2,4); plot(lambda4/sum(lambda4)*100,'bo-','LineWidth',[2]); ylabel('Variance Captured
(%)'); xlabel('Eigenvalue'); title('CaseN\_4');
155.
156. %% Plot of Principal Components
157.
158. figure(4);
159. subplot(2,2,1); plot(PC_1_mean(1,:), 'bo-', 'LineWidth',[2]); ylabel('Principal Component');
xlabel('Variable'); title('CaseN\_1'); set(gca,'FontSize',20); axis([1 6 -500 600]);
160. hold on; plot(PC_1_mean(2,:),'ro-', 'LineWidth',[2]);
161. subplot(2,2,2); plot(PC_2_mean(1,:), 'bo-', 'LineWidth',[2]); ylabel('Principal Component');
xlabel('Variable'); title('CaseN\_2'); set(gca,'FontSize',20); axis([1 6 -900 600]);
162. hold on; plot(PC_2_mean(2,:),'ro-', 'LineWidth',[2]);
163. subplot(2,2,3); plot(PC_3_mean(1,:), 'bo-', 'LineWidth',[2]); ylabel('Principal Component');
xlabel('Variable'); title('CaseN\_3'); set(gca,'FontSize',20); axis([1 6 -500 500]);
164. hold on; plot(PC_3_mean(2,:),'ro-', 'LineWidth',[2]);
165. subplot(2,2,4); plot(PC_4_mean(1,:), 'bo-', 'LineWidth',[2]); ylabel('Principal Component');
xlabel('Variable'); title('CaseN\_4'); set(gca,'FontSize',20); axis([1 6 -700 700]);
166. hold on; plot(PC_4_mean(2,:),'ro-', 'LineWidth',[2]);
167.
168. %% Tracking paint can position
169. function [x,y] = trackPosition(video,percent)
170.     frameCount = size(video,4);
171.
172.     for i = 1:frameCount
173.         minIntensity = percent*max(max(rgb2gray(video(:,:,:,i))));
174.         [x_max,y_max] = find(rgb2gray(video(:,:,:,i))>minIntensity);
175.         L = [x_max,y_max];
176.         [idx,c] = kmeans(L,1);
177.         x(i) = c(1,1);
178.         y(i) = c(1,2);
179.     end
180.       x = x - mean(x);
181.       y = y - mean(y);
182. end
```