Motion characterization of a spring-mass system with principal component analysis

Arman Naderi

Introduction and Overview:

Principal component analysis (PCA) is a statistical technique by which variables that may or may not be correlated are transformed into principal components - a set of values of linearly independent variables. This technique is a vital tool in investigative data analysis because it identifies an uncorrelated orthogonal basis set of vectors across which the dataset has the greatest variance. By identifying these vectors, the dimensionality of the dataset can be reduced, thus revealing the most relevant information for the characteristic relationships between variables in the dataset. In this report, I utilize PCA to characterize the mechanical behavior of a spring-mass system through video recordings.

The system under analysis is a white paint bucket attached to a spring, wherein a point source of light is attached to the top surface of the paint bucket to assist with tracking its movement. There are three different camera angles recorded during four different test cases that are applied to the spring-mass system. The first test applied is the ideal case for the system in which the mass's displacement is recorded when the mass is dropped along a single axis, with no external horizontal displacement or rotation applied. The second test is the same as the ideal case for the spring-mass system wherein no external horizontal displacement or rotation is applied, however, camera shake is introduced to add noise to the recording of the system. The third test case adds horizontal displacement to the dropped mass, so the resulting behavior involves movement in all three principal axes. The fourth test case adds horizontal displacement and rotation to the dropped mass, so the resulting behavior will also involve movement in all three principal axes. The video data recorded will inherently have redundant information and noise, and thus PCA will be applied to extract meaningful behavior from the system.

Prior to beginning data analysis, data must be collected. In this case, the motion of the paint bucket must be characterized for each of the twelve recorded videos. To do this, I have applied a simple image processing algorithm which utilizes image binarization and masking to isolates the location of a single pixel that represents the location of the paint bucket. The coordinates of the paint bucket for each of the camera angles, for each of the tests, were concatenated and then used for PCA. I carried out PCA by using singular value decomposition (SVD).

Theoretical Background:

Singular value decomposition is the factorization of a mxn matrix, A, into orthogonal/orthonormal matrices that cause rotation, coordinate scaling, and another rotation of A. The full SVD of A is defined as:

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

wherein U is a mxm matrix, Σ is a mxn diagonal matrix, V is a nxn diagonal matrix. In order to perform a full SVD, columns or rows of zeros are added to the U and V* matrices to make them unitary. The matrix U contains the left singular vectors of A and the matrix V* contains the right singular vectors of A. The matrix Σ contains r (rank) positive diagonal singular values (σ_j), where they are ordered in descending value by convention.

The SVD is computed via the following:

$$A^TA = (U\Sigma V^*)^T(U\Sigma V^*) \quad (2.a)$$
$$= V\Sigma^2V^* \quad (2.b)$$

$$AA^T = (U\Sigma V^*)(U\Sigma V^*)^T \quad (3.a)$$
$$= U\Sigma^2U^* \quad (3.c)$$

$$A^TAV = V\Sigma^2 \quad (4.a)$$
$$AA^TU = U\Sigma^2. \quad (4.b)$$

The orthonormal basis vectors for U and V are produced by computing the normalized eigenvectors of 4.a and 4.b. By taking the square roots of the eigenvalues, the singular values are computed. This decomposition of matrix A allows for its lower dimensional representation, which can be useful for reducing noisy data or redundant information.

Algorithm Implementation and Development:

For each of the camera angles, for each of the applied tests, an image processing algorithm was implemented that isolated a single pixel location that represented the location of the bucket for each frame. First, the algorithm converted the RGB image of the frame to grayscale. Then, the algorithm converted the grayscale image into a logical matrix of ones and zeros based on whether or not each pixel in the image met a certain threshold value. Then, for each frame, an average pixel location was calculated using the average of the indices of the pixels in the binarized image that were not equal to zero. Simultaneous to this was the implementation of mask which simply excluded features in the recorded video that were not relevant to the spring-mass system. By adjusting the threshold used for recording the position of the bucket, and the mask around the region of interest for the video, the x and y position of the bucket was recorded in a matrix with a length equal to the number of frames in the video.

After the positions were recorded, the data had to be massaged so that it could be used for PCA. First, the data was separated into x and y vectors. Then, in order to ensure that the videos all started at the same time, the minimum y position of the first and second camera angle, and the maximum x position of the third camera angle were used to initialize a starting point for all the matrices. After, in order to ensure that each of the camera angles had the same length of recorded observations, the length of all the x and y vectors were curtailed to the minimum length from the three recorded camera angles, for each of the applied tests. Then, the average x and y coordinate was subtracted from the x and y vectors in order to ensure that the SVD did not weight the absolute location of a datapoint within a given data set. The x and y vectors for each camera angle were concatenated into a single matrix, and then an economical SVD was performed on that matrix. To find the energy of each of the singular values, each singular value was divided by the singular value average. Then, the displacement of the bucket, the relative energy of the singular values, and the data's projection onto the principal components were plotted for each of the tests.

Computational Results:

The computational results for each of the camera angles on test 1, the ideal case is shown in Figure 1 below. Figure 1 demonstrates that the motion of the bucket's displacement can be captured primarily with one principal component. The singular value's relative energies are: SV 1 ~ 0.87, SV 2 ~ 0.08, SV 3 ~ 0.04. This makes sense as this video was recorded with the intention of being the ideal case with no noise resulting from experimental error (horizontal displacement or rotation) or from observational error (camera shake). The primary reason why there are features registered in the 2nd and 3rd singular values is noise accumulated from the image processing step of the algorithm. By further refining the algorithm to be less susceptible to sudden lighting changes in the frame, I will be able to more accurately record the position of the bucket as it changes with time.

The computational results for each of the camera angles on test 2, the ideal case with camera shake is shown in Figure 2 below. Figure 2 demonstrates that the motion of the bucket's displacement can be captured in one principal component, but with a significant reduction in quality in that more features are resolved in a variety of other principal components. The singular value's relative energies are: SV 1 ~ 0.64, SV 2 ~ 0.17, SV 3 ~ 0.10, SV 4 ~ 0.04. This significant increase in distribution of features across singular values makes sense as this video was recorded with the intention of introducing noise to the recording of the spring-mass system thus resulting in observational error. This noise, alongside by image processing algorithm's poor ability to resolve sudden light changes, is what causes the behavior of the spring-mass system to appear less uniform.

The computational results for each of the camera angles on test 3, which adds horizontal displacement is shown in Figure 3 below. Figure 3 demonstrates that the motion of the bucket's displacement can be captured in two primary principal components. The singular value's relative energies are: SV 1 ~ 0.56, SV 2 ~ 0.29, SV 3 ~ .12, SV 4 ~ 0.03. The increase in distribution of features in the second singular values makes sense as this video was recorded with the intention of introducing motion in a direction that is not natural to the spring-mass system. The PCA captured the motion that was orthogonal to the primary axis of movement; however, noise was still captured in the third principal component.

The computational results for each of the camera angles on test 4, which adds rotation to horizontal displacement is shown in Figure 4 below. Figure 4 demonstrates that the motion of the bucket's displacement can be captured in two primary principal components. The singular value's relative energies are: SV 1 ~ 0.81, SV 2 ~ 0.11, SV 3 ~ .05, SV 4 ~ 0.02. The increase in distribution of features in the second singular values makes sense as this video was recorded with the intention of introducing motion in a direction that is not natural to the spring-mass system. The PCA captured the motion that was orthogonal to the primary axis of movement; however, noise was still captured in the third principal component. It can be concluded that rotation does little to influence the behavior of this oscillatory system.

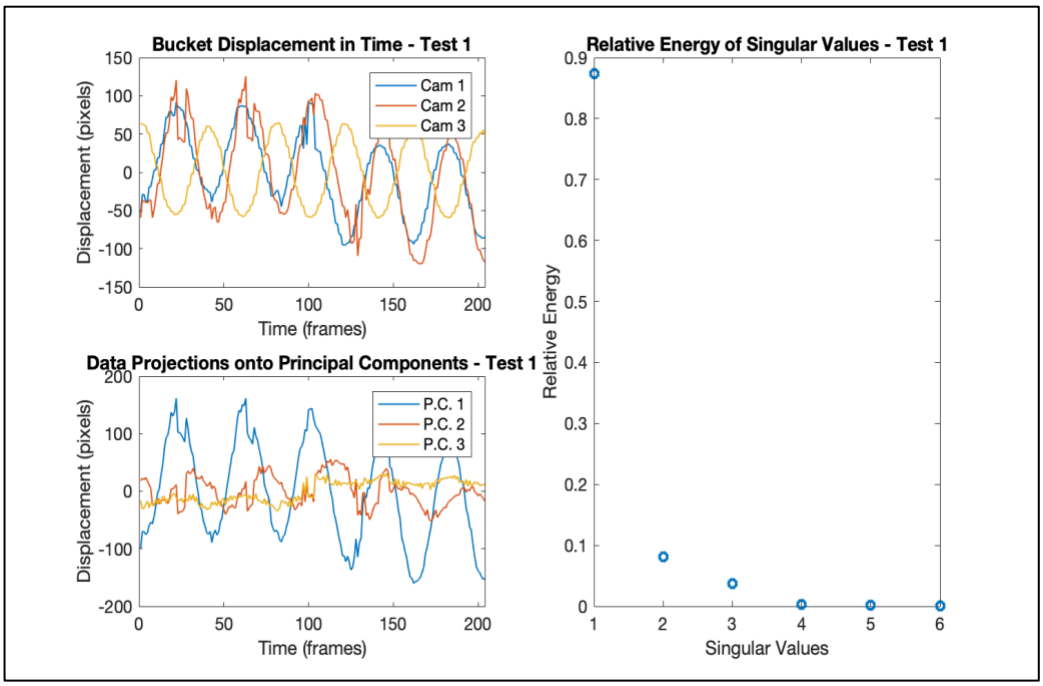

Figure 1: (top left) A plot of the bucket displacement over time from camera angles 1, 2, and 3. (bottom left) A plot of the data projections onto the principal components (P.C.). (right) A plot of the relative energies of the singular values.

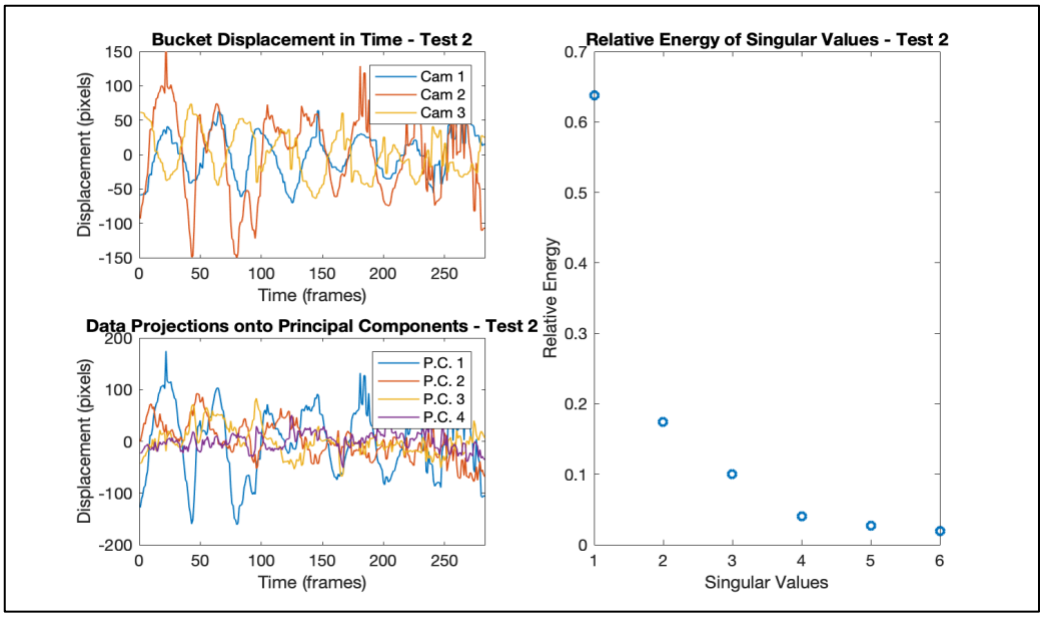

Figure 2: (top left) A plot of the bucket displacement over time from camera angles 1, 2, and 3. (bottom left) A plot of the data projections onto the principal components (P.C.). (right) A plot of the relative energies of the singular values.

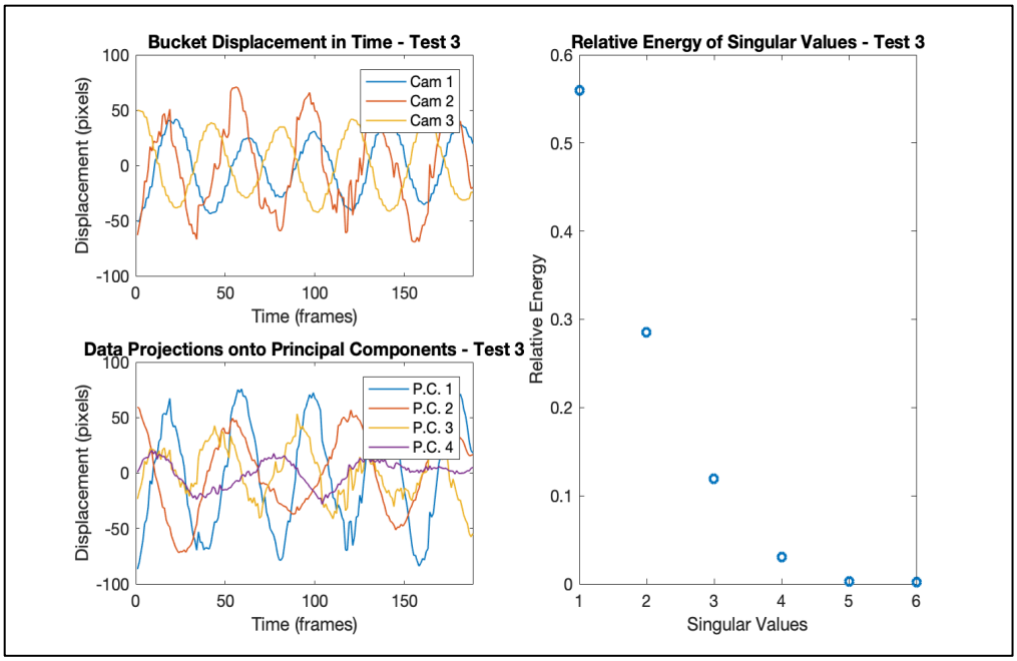

Figure 3: (top left) A plot of the bucket displacement over time from camera angles 1, 2, and 3. (bottom left) A plot of the data projections onto the principal components (P.C.). (right) A plot of the relative energies of the singular values.

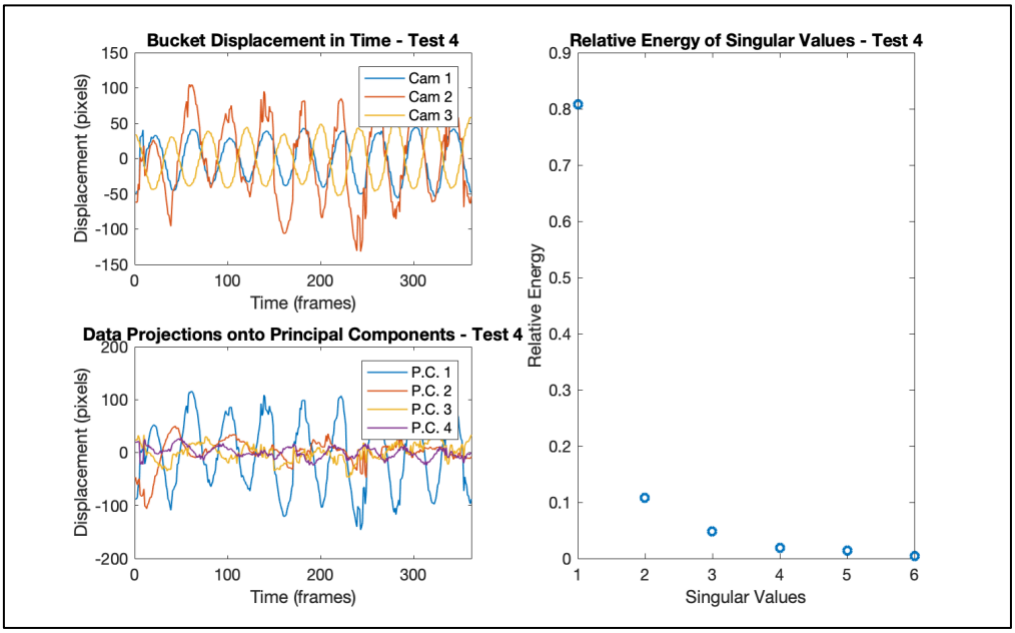

Figure 4: (top left) A plot of the bucket displacement over time from camera angles 1, 2, and 3. (bottom left) A plot of the data projections onto the principal components (P.C.). (right) A plot of the relative energies of the singular values.

Summary and Conclusions:

By utilizing PCA, I was able to analyze the oscillatory mechanics of the spring-mass system under a variety of test conditions. It is important to note that PCA itself analyzes the effects of the image processing of the algorithm as well as the mechanics under study. Through utilizing PCA, I was able to reduce the redundancy of information in my system, and decrease noise from interfering variables. This method for data analysis makes it easy for quickly identifying which variables are most closely related.

Appendix A:

I = rgb2gray(RGB); Converts a color image RGB to grayscale image, I, by maintaining luminance but eliminating hue and saturation information.
BW = Imbinarize(I); Creates a logical, binary image, BW, from a 2D or 3D grayscale image, I, by using a threshold to determine which value(s) will be set to 1s or 0s.
K = find(x); Returns a column vector containing the linear indices of each nonzero element in array X.
[row col] = ind2sub(sz, ind); Returns the arrays, row and col, containing the respective row and column subscripts corresponding to the linear indices, ind, for a matrix of size, sz.
[U, S, V] = svd(A, 'econ') = Returns an economy-size decomposition of an m-by-n matrix A. The economy-size decomposition removes the extra rows or columns of zeros from the diagonal matrix of singular values, S, and their corresponding columns in U or V. This can improve execution time and reduce storage requirements.
M = mean(A); Returns the mean of the elements of A along the first array dimension whose size does not equal 1.
X = zeros(sz1, … , szN); Returns an array of zeros size, sz1-by-…-by-szN. Commonly used to pre-allocate memory for variable or matrix.
[M I] = min(___); Returns the minimum value, M, and its index, I, in an array.
[M I] = max(___); Returns the maximum value, M, and its index, I, in an array.
D = diag(v); Returns a square diagonal matrix with the elements of vector, v, on the main diagonal.

Appendix B:

```
%% Test 1 - Ideal Case: Image Processing
clear all; close all; clc;

% cam1_1
load('cam1_1.mat')

numFrames = size(vidFrames1_1,4);

y_mask = 180:480;
x_mask = 300:345;
a_pos1_1 = zeros(numFrames,2);
for j = 1:numFrames
    RGB = vidFrames1_1(y_mask,x_mask,:,j);
    gray = rgb2gray(RGB);
    gray_b = imbinarize(gray, 0.95);
    w_pixel = find(gray_b);
    sum_pos = [0,0];
    for jj = (1:length(w_pixel))
        [y,x] = ind2sub(size(gray_b), w_pixel(jj));
    end
    a_pos1_1(j,:) = [mean(x),mean(y)];
%     imshow(gray_b)
%     hold on
%     plot(a_pos1_1(:,1), a_pos1_1(:,2),'.r','Markersize', 10)
%     drawnow
end
```

```
% cam2_1
load('cam2_1.mat')

numFrames = size(vidFrames2_1,4);

y_mask = 80:380;
x_mask = 250:340;
a_pos2_1 = zeros(numFrames,2);

for j = 1:numFrames
    RGB = vidFrames2_1(y_mask,x_mask,:,j);
    gray = rgb2gray(RGB);
    gray_b = imbinarize(gray, 0.965);
    w_pixel = find(gray_b);
    sum_pos = [0,0];
    for jj = (1:length(w_pixel))
        [y,x] = ind2sub(size(gray_b), w_pixel(jj));
    end
    a_pos2_1(j,:) = [mean(x),mean(y)];
%     imshow(gray_b)
%     hold on
%     plot(a_pos2_1(:,1), a_pos2_1(:,2),'.r','Markersize', 10)
%     drawnow
end

% cam3_1
load('cam3_1.mat')

numFrames = size(vidFrames3_1,4);

y_mask = 210:330;
x_mask = 280:480;
a_pos3_1 = zeros(numFrames,2);

for j = 1:numFrames
    RGB = vidFrames3_1(y_mask,x_mask,:,j);
    gray = rgb2gray(RGB);
    gray_b = imbinarize(gray, 0.9);
    w_pixel = find(gray_b);
    sum_pos = [0,0];
    for jj = (1:length(w_pixel))
        [y,x] = ind2sub(size(gray_b), w_pixel(jj));
    end
    a_pos3_1(j,:) = [mean(x),mean(y)];
%     imshow(gray_b)
%     hold on
%     plot(a_pos3_1(:,1), a_pos3_1(:,2),'.r','Markersize', 10)
%     drawnow
end

%% Test 1 - Ideal Case: PCA
% Separate x and y components from position matrix
x1_1 = a_pos1_1(:, 1); y1_1 = a_pos1_1(:, 2);
x2_1 = a_pos2_1(:, 1); y2_1 = a_pos2_1(:, 2);
x3_1 = a_pos3_1(:, 1); y3_1 = a_pos3_1(:, 2);
```

```
% Make the initial start the same
[~,y1_1_min] = min(y1_1(1:50));
[~,y2_1_min] = min(y2_1(1:50));
[~,x3_1_max] = max(x3_1(1:50));

x1_1 = x1_1(y1_1_min:end); y1_1 = y1_1(y1_1_min:end);
x2_1 = x2_1(y2_1_min:end); y2_1 = y2_1(y2_1_min:end);
x3_1 = x3_1(x3_1_max:end); y3_1 = y3_1(x3_1_max:end);

% Make number of snapshots equal to the minimum
min_snap = min([length(x1_1), length(x2_1), length(x3_1)]);

x1_1 = x1_1(1:min_snap); y1_1 = y1_1(1:min_snap);
x2_1 = x2_1(1:min_snap); y2_1 = y2_1(1:min_snap);
x3_1 = x3_1(1:min_snap); y3_1 = y3_1(1:min_snap);

% Find the average value of the x and y coordinate for each camera to
% prevent the SVD from weighting absolute location
a_x1_1 = mean(x1_1); a_y1_1 = mean(y1_1);
a_x2_1 = mean(x2_1); a_y2_1 = mean(y2_1);
a_x3_1 = mean(x3_1); a_y3_1 = mean(y3_1);

% Subtract each row element from its corresponding average
for n = 1:length(x1_1)
    x1_1(n) = x1_1(n) - a_x1_1; y1_1(n) = y1_1(n) - a_y1_1;
    x2_1(n) = x2_1(n) - a_x2_1; y2_1(n) = y2_1(n) - a_y2_1;
    x3_1(n) = x3_1(n) - a_x3_1; y3_1(n) = y3_1(n) - a_y3_1;
end

% Variables and snapshots collated into a single matrix
A = [x1_1';y1_1';x2_1';y2_1';x3_1';y3_1'];

% Take SVD of collated matrix withan economy decomposition
[U,S,V] = svd(A,'econ');

% Find the energy of the singular values by dividing each singular
% value by the average of all of the singular values
S_col = diag(S);

for j = 1:length(S_col)
    S_col(j,1) = S_col(j,1)*S_col(j,1);
end

S_sum = sum(S_col);
for j = 1:length(S_col)
    S_col(j,1) = S_col(j,1)/S_sum;
end
S_diag = diag(S_col);

% Singular value energy plot
figure(1)
subplot(2,2,[2, 4])
plot(1:length(S_col), S_col, 'o', 'LineWidth', 2)
```

```
set(gca,'Fontsize',12)
xlabel('Singular Values');
ylabel('Relative Energy');
title('Relative Energy of Singular Values - Test 1');

% plot of all camera angles showing major variation direction - z
subplot(2,2,1)
plot(1:length(y1_1),y1_1,1:length(y1_1),y2_1,1:length(y1_1),x3_1,'LineWidth',
1)
set(gca,'Xlim',[0,length(y1_1)],'Ylim',[-150,150],'Fontsize',12)
legend("Cam 1","Cam 2","Cam 3")
xlabel('Time (frames)');
ylabel('Displacement (pixels)');
title('Bucket Displacement in Time - Test 1');

% plot of data projections onto pricipal components
subplot(2,2,3)
A_proj = U'*A;
plot(1:length(A_proj(1,:)), A_proj(1,:),'LineWidth',1)
hold on
plot(1:length(A_proj(2,:)), A_proj(2,:),'LineWidth',1)
hold on
plot(1:length(A_proj(3,:)), A_proj(3,:),'LineWidth',1)
hold on
legend("P.C. 1","P.C. 2","P.C. 3")
set(gca,'Xlim',[0,length(A_proj(1,:))],'Fontsize',12)
xlabel('Time (frames)');
ylabel('Displacement (pixels)');
title('Data Projections onto Principal Components - Test 1');

%% Test 2 - Noisy Case: Image processing
clear all; close all; clc;

% cam1_2
load('cam1_2.mat')

numFrames = size(vidFrames1_2,4);

y_mask = 180:480;
x_mask = 280:400;
a_pos1_2 = zeros(numFrames,2);
for j = 1:numFrames
    RGB = vidFrames1_2((y_mask),(x_mask),:,j);
    gray = rgb2gray(RGB);
    gray_b = imbinarize(gray, 0.9999999999);
    w_pixel = find(gray_b);
    sum_pos = [0,0];
    for jj = (1:length(w_pixel))
        [y,x] = ind2sub(size(gray_b), w_pixel(jj));
    end
    a_pos1_2(j,:) = [mean(x),mean(y)];
%     imshow(gray_b)
%     hold on
%     plot(a_pos1_2(:,1), a_pos1_2(:,2),'.r','Markersize', 10)
%     drawnow
end
```

```
% cam2_2
load('cam2_2.mat')

numFrames = size(vidFrames2_2,4);

y_mask = 40:420;
x_mask = 200:460;
a_pos2_2 = zeros(numFrames,2);
for j = 1:numFrames
    RGB = vidFrames2_2((y_mask),(x_mask),:,j);
    gray = rgb2gray(RGB);
    gray_b = imbinarize(gray, 0.95);
    w_pixel = find(gray_b);
    sum_pos = [0,0];
    for jj = (1:length(w_pixel))
        [y,x] = ind2sub(size(gray_b), w_pixel(jj));
    end
    a_pos2_2(j,:) = [mean(x),mean(y)];
%     imshow(gray_b)
%     hold on
%     plot(a_pos2_2(:,1), a_pos2_2(:,2),'.r','Markersize', 10)
%     drawnow
end

% cam3_2
load('cam3_2.mat')

numFrames = size(vidFrames3_2,4);

y_mask = 200:320;
x_mask = 300:500;
a_pos3_2 = zeros(numFrames,2);
for j = 1:numFrames
    RGB = vidFrames3_2((y_mask),(x_mask),:,j);
    gray = rgb2gray(RGB);
    gray_b = imbinarize(gray, 0.95);
    w_pixel = find(gray_b);
    sum_pos = [0,0];
    for jj = (1:length(w_pixel))
        [y,x] = ind2sub(size(gray_b), w_pixel(jj));
    end
    a_pos3_2(j,:) = [mean(x),mean(y)];
%     imshow(gray_b)
%     hold on
%     plot(a_pos3_2(:,1), a_pos3_2(:,2),'.r','Markersize', 10)
%     drawnow
end

%% Test 2 - Noisy Case: PCA
% Separate x and y components from position matrix
x1_2 = a_pos1_2(:, 1); y1_2 = a_pos1_2(:, 2);
x2_2 = a_pos2_2(:, 1); y2_2 = a_pos2_2(:, 2);
x3_2 = a_pos3_2(:, 1); y3_2 = a_pos3_2(:, 2);
```

```
% Make the initial start the same
[~,y1_2_min] = min(y1_2(1:50));
[~,y2_2_min] = min(y2_2(1:50));
[~,x3_2_max] = max(x3_2(1:50));

x1_2 = x1_2(y1_2_min:end); y1_2 = y1_2(y1_2_min:end);
x2_2 = x2_2(y2_2_min:end); y2_2 = y2_2(y2_2_min:end);
x3_2 = x3_2(x3_2_max:end); y3_2 = y3_2(x3_2_max:end);

% Make number of snapshots equal to the minimum
min_snap = min([length(x1_2), length(x2_2), length(x3_2)]);

x1_2 = x1_2(1:min_snap); y1_2 = y1_2(1:min_snap);
x2_2 = x2_2(1:min_snap); y2_2 = y2_2(1:min_snap);
x3_2 = x3_2(1:min_snap); y3_2 = y3_2(1:min_snap);

% Find the average value of the x and y coordinate for each camera to
% prevent the SVD from weighting absolute location
a_x1_2 = mean(x1_2); a_y1_2 = mean(y1_2);
a_x2_2 = mean(x2_2); a_y2_2 = mean(y2_2);
a_x3_2 = mean(x3_2); a_y3_2 = mean(y3_2);

% Subtract each row element from its corresponding average
for n = 1:length(x1_2)
    x1_2(n) = x1_2(n) - a_x1_2; y1_2(n) = y1_2(n) - a_y1_2;
    x2_2(n) = x2_2(n) - a_x2_2; y2_2(n) = y2_2(n) - a_y2_2;
    x3_2(n) = x3_2(n) - a_x3_2; y3_2(n) = y3_2(n) - a_y3_2;
end

% Variables and snapshots collated into a single matrix
A = [x1_2';y1_2';x2_2';y2_2';x3_2';y3_2'];

% Take SVD of collated matrix withan economy decomposition
[U,S,V] = svd(A,'econ');

% Find the energy of the singular values by dividing each singular
% value by the average of all of the singular values
S_col = diag(S);

for j = 1:length(S_col)
    S_col(j,1) = S_col(j,1)*S_col(j,1);
end

S_sum = sum(S_col);
for j = 1:length(S_col)
    S_col(j,1) = S_col(j,1)/S_sum;
end
S_diag = diag(S_col);

% Singular value energy plot
figure(1)
subplot(2,2,[2, 4])
plot(1:length(S_col), S_col, 'o', 'LineWidth', 2)
set(gca,'Fontsize',12)
xlabel('Singular Values');
```

```
ylabel('Relative Energy');
title('Relative Energy of Singular Values - Test 2');

% plot of all camera angles showing major variation direction - z
subplot(2,2,1)
plot(1:length(y1_2),y1_2,1:length(y1_2),y2_2,1:length(y1_2),x3_2,'LineWidth',
1)
set(gca,'Xlim',[0,length(y1_2)],'Ylim',[-150,150],'Fontsize',12)
legend("Cam 1","Cam 2","Cam 3")
xlabel('Time (frames)');
ylabel('Displacement (pixels)');
title('Bucket Displacement in Time - Test 2');

% plot of data projections onto pricipal components
subplot(2,2,3)
A_proj = U'*A;
plot(1:length(A_proj(1,:)), A_proj(1,:),'LineWidth',1)
hold on
plot(1:length(A_proj(2,:)), A_proj(2,:),'LineWidth',1)
hold on
plot(1:length(A_proj(3,:)), A_proj(3,:),'LineWidth',1)
hold on
plot(1:length(A_proj(4,:)), A_proj(4,:),'LineWidth',1)
hold on
legend("P.C. 1","P.C. 2","P.C. 3","P.C. 4")
set(gca,'Xlim',[0,length(A_proj(1,:))],'Fontsize',12)
xlabel('Time (frames)');
ylabel('Displacement (pixels)');
title('Data Projections onto Principal Components - Test 2');

%% Test 3 - Horizontal Displacement: Image processing
clear all; close all; clc;

% cam1_3
load('cam1_3.mat')

numFrames = size(vidFrames1_3,4);

y_mask = 220:480;
x_mask = 280:400;
a_pos1_3 = zeros(numFrames,2);
for j = 1:numFrames
    RGB = vidFrames1_3((y_mask),(x_mask),:,j);
    gray = rgb2gray(RGB);
    gray_b = imbinarize(gray, 0.85);
    w_pixel = find(gray_b);
    sum_pos = [0,0];
    for jj = (1:length(w_pixel))
        [y,x] = ind2sub(size(gray_b), w_pixel(jj));
    end
    a_pos1_3(j,:) = [mean(x),mean(y)];
%     imshow(gray_b)
%     hold on
%     plot(a_pos1_3(:,1), a_pos1_3(:,2),'.r','Markersize', 10)
%     drawnow
end
```

```
% cam2_3
load('cam2_3.mat')

numFrames = size(vidFrames2_3,4);

y_mask = 180:420;
x_mask = 180:460;
a_pos2_3 = zeros(numFrames,2);
for j = 1:numFrames
    RGB = vidFrames2_3((y_mask),(x_mask),:,j);
    gray = rgb2gray(RGB);
    gray_b = imbinarize(gray, 0.985);
    w_pixel = find(gray_b);
    sum_pos = [0,0];
    for jj = (1:length(w_pixel))
        [y,x] = ind2sub(size(gray_b), w_pixel(jj));
    end
    a_pos2_3(j,:) = [mean(x),mean(y)];
%     imshow(gray_b)
%     hold on
%     plot(a_pos2_3(:,1), a_pos2_3(:,2),'.r','Markersize', 10)
%     drawnow
end

% cam3_3
load('cam3_3.mat')

numFrames = size(vidFrames3_3,4);

y_mask = 200:320;
x_mask = 300:500;
a_pos3_3 = zeros(numFrames,2);
for j = 1:numFrames
    RGB = vidFrames3_3((y_mask),(x_mask),:,j);
    gray = rgb2gray(RGB);
    gray_b = imbinarize(gray, 0.95);
    w_pixel = find(gray_b);
    sum_pos = [0,0];
    for jj = (1:length(w_pixel))
        [y,x] = ind2sub(size(gray_b), w_pixel(jj));
    end
    a_pos3_3(j,:) = [mean(x),mean(y)];
%     imshow(gray_b)
%     hold on
%     plot(a_pos3_3(:,1), a_pos3_3(:,2),'.r','Markersize', 10)
%     drawnow
end

%% Test 3 - Horizontal displacement: PCA
% Separate x and y components from position matrix
x1_3 = a_pos1_3(:, 1); y1_3 = a_pos1_3(:, 2);
x2_3 = a_pos2_3(:, 1); y2_3 = a_pos2_3(:, 2);
x3_3 = a_pos3_3(:, 1); y3_3 = a_pos3_3(:, 2);
```

```
% Make the initial start the same
[~,y1_3_min] = min(y1_3(1:50));
[~,y2_3_min] = min(y2_3(1:50));
[~,x3_3_max] = max(x3_3(1:50));

x1_3 = x1_3(y1_3_min:end); y1_3 = y1_3(y1_3_min:end);
x2_3 = x2_3(y2_3_min:end); y2_3 = y2_3(y2_3_min:end);
x3_3 = x3_3(x3_3_max:end); y3_3 = y3_3(x3_3_max:end);

% Make number of snapshots equal to the minimum
min_snap = min([length(x1_3), length(x2_3), length(x3_3)]);

x1_3 = x1_3(1:min_snap); y1_3 = y1_3(1:min_snap);
x2_3 = x2_3(1:min_snap); y2_3 = y2_3(1:min_snap);
x3_3 = x3_3(1:min_snap); y3_3 = y3_3(1:min_snap);

% Find the average value of the x and y coordinate for each camera to
% prevent the SVD from weighting absolute location
a_x1_3 = mean(x1_3); a_y1_3 = mean(y1_3);
a_x2_3 = mean(x2_3); a_y2_3 = mean(y2_3);
a_x3_3 = mean(x3_3); a_y3_3 = mean(y3_3);

% Subtract each row element from its corresponding average
for n = 1:length(x1_3)
    x1_3(n) = x1_3(n) - a_x1_3; y1_3(n) = y1_3(n) - a_y1_3;
    x2_3(n) = x2_3(n) - a_x2_3; y2_3(n) = y2_3(n) - a_y2_3;
    x3_3(n) = x3_3(n) - a_x3_3; y3_3(n) = y3_3(n) - a_y3_3;
end

% Variables and snapshots collated into a single matrix
A = [x1_3';y1_3';x2_3';y2_3';x3_3';y3_3'];

% Take SVD of collated matrix withan economy decomposition
[U,S,V] = svd(A,'econ');

% Find the energy of the singular values by dividing each singular
% value by the average of all of the singular values
S_col = diag(S);

for j = 1:length(S_col)
    S_col(j,1) = S_col(j,1)*S_col(j,1);
end

S_sum = sum(S_col);
for j = 1:length(S_col)
    S_col(j,1) = S_col(j,1)/S_sum;
end
S_diag = diag(S_col);

% Singular value energy plot
figure(1)
subplot(2,2,[2, 4])
plot(1:length(S_col), S_col, 'o', 'LineWidth', 2)
set(gca,'Fontsize',12)
xlabel('Singular Values');
```

```
ylabel('Relative Energy');
title('Relative Energy of Singular Values - Test 3');

% plot of all camera angles showing major variation direction - z
subplot(2,2,1)
plot(1:length(y1_3),y1_3,1:length(y1_3),y2_3,1:length(y1_3),x3_3,'LineWidth',
1)
set(gca,'Xlim',[0,length(y1_3)],'Ylim',[-100,100],'Fontsize',12)
legend("Cam 1","Cam 2","Cam 3")
xlabel('Time (frames)');
ylabel('Displacement (pixels)');
title('Bucket Displacement in Time - Test 3');

% plot of data projections onto pricipal components
subplot(2,2,3)
A_proj = U'*A;
plot(1:length(A_proj(1,:)), A_proj(1,:),'LineWidth',1)
hold on
plot(1:length(A_proj(2,:)), A_proj(2,:),'LineWidth',1)
hold on
plot(1:length(A_proj(3,:)), A_proj(3,:),'LineWidth',1)
hold on
plot(1:length(A_proj(4,:)), A_proj(4,:),'LineWidth',1)
hold on
legend("P.C. 1","P.C. 2","P.C. 3","P.C. 4")
set(gca,'Xlim',[0,length(A_proj(1,:))],'Fontsize',12)
xlabel('Time (frames)');
ylabel('Displacement (pixels)');
title('Data Projections onto Principal Components - Test 3');

%% Test 4 - Horizontal Displacement and Rotation: Image processing
clear all; close all; clc;

% cam1_4
load('cam1_4.mat')

numFrames = size(vidFrames1_4,4);

y_mask = 230:440;
x_mask = 300:440;
a_pos1_4 = zeros(numFrames,2);
for j = 1:numFrames
    RGB = vidFrames1_4((y_mask),(x_mask),:,j);
    gray = rgb2gray(RGB);
    gray_b = imbinarize(gray, 0.675);
    w_pixel = find(gray_b);
    sum_pos = [0,0];
    for jj = (1:length(w_pixel))
        [y,x] = ind2sub(size(gray_b), w_pixel(jj));
    end
    a_pos1_4(j,:) = [mean(x),mean(y)];
%     imshow(gray_b)
%     hold on
%     plot(a_pos1_4(:,1), a_pos1_4(:,2),'.r','Markersize', 10)
%     drawnow
end
```

```
% cam2_4
load('cam2_4.mat')

numFrames = size(vidFrames2_4,4);

y_mask = 100:360;
x_mask = 200:450;
a_pos2_4 = zeros(numFrames,2);
for j = 1:numFrames
    RGB = vidFrames2_4((y_mask),(x_mask),:,j);
    gray = rgb2gray(RGB);
    gray_b = imbinarize(gray, 0.985);
    w_pixel = find(gray_b);
    sum_pos = [0,0];
    for jj = (1:length(w_pixel))
        [y,x] = ind2sub(size(gray_b), w_pixel(jj));
    end
    a_pos2_4(j,:) = [mean(x),mean(y)];
%     imshow(gray_b)
%     hold on
%     plot(a_pos2_4(:,1), a_pos2_4(:,2),'.r','Markersize', 10)
%     drawnow
end

% cam3_4
load('cam3_4.mat')

numFrames = size(vidFrames3_4,4);

y_mask = 120:300;
x_mask = 320:520;
a_pos3_4 = zeros(numFrames,2);
for j = 1:numFrames
    RGB = vidFrames3_4((y_mask),(x_mask),:,j);
    gray = rgb2gray(RGB);
    gray_b = imbinarize(gray, 0.9);
    w_pixel = find(gray_b);
    sum_pos = [0,0];
    for jj = (1:length(w_pixel))
        [y,x] = ind2sub(size(gray_b), w_pixel(jj));
    end
    a_pos3_4(j,:) = [mean(x),mean(y)];
%     imshow(gray_b)
%     hold on
%     plot(a_pos3_4(:,1), a_pos3_4(:,2),'.r','Markersize', 10)
%     drawnow
end

%% Test 4 - Horizontal displacement and Rotation: PCA
% Separate x and y components from position matrix
x1_4 = a_pos1_4(:, 1); y1_4 = a_pos1_4(:, 2);
x2_4 = a_pos2_4(:, 1); y2_4 = a_pos2_4(:, 2);
x3_4 = a_pos3_4(:, 1); y3_4 = a_pos3_4(:, 2);
```

```
% Make the initial start the same
[~,y1_4_min] = min(y1_4(1:50));
[~,y2_4_min] = min(y2_4(1:50));
[~,x3_4_max] = max(x3_4(1:50));

x1_4 = x1_4(y1_4_min:end); y1_4 = y1_4(y1_4_min:end);
x2_4 = x2_4(y2_4_min:end); y2_4 = y2_4(y2_4_min:end);
x3_4 = x3_4(x3_4_max:end); y3_4 = y3_4(x3_4_max:end);

% Make number of snapshots equal to the minimum
min_snap = min([length(x1_4), length(x2_4), length(x3_4)]);

x1_4 = x1_4(1:min_snap); y1_4 = y1_4(1:min_snap);
x2_4 = x2_4(1:min_snap); y2_4 = y2_4(1:min_snap);
x3_4 = x3_4(1:min_snap); y3_4 = y3_4(1:min_snap);

% Find the average value of the x and y coordinate for each camera to
% prevent the SVD from weighting absolute location
a_x1_4 = mean(x1_4); a_y1_4 = mean(y1_4);
a_x2_4 = mean(x2_4); a_y2_4 = mean(y2_4);
a_x3_4 = mean(x3_4); a_y3_4 = mean(y3_4);

% Subtract each row element from its corresponding average
for n = 1:length(x1_4)
    x1_4(n) = x1_4(n) - a_x1_4; y1_4(n) = y1_4(n) - a_y1_4;
    x2_4(n) = x2_4(n) - a_x2_4; y2_4(n) = y2_4(n) - a_y2_4;
    x3_4(n) = x3_4(n) - a_x3_4; y3_4(n) = y3_4(n) - a_y3_4;
end

% Variables and snapshots collated into a single matrix
A = [x1_4';y1_4';x2_4';y2_4';x3_4';y3_4'];

% Take SVD of collated matrix withan economy decomposition
[U,S,V] = svd(A,'econ');

% Find the energy of the singular values by dividing each singular
% value by the average of all of the singular values
S_col = diag(S);

for j = 1:length(S_col)
    S_col(j,1) = S_col(j,1)*S_col(j,1);
end

S_sum = sum(S_col);
for j = 1:length(S_col)
    S_col(j,1) = S_col(j,1)/S_sum;
end
S_diag = diag(S_col);

% Singular value energy plot
figure(1)
subplot(2,2,[2, 4])
plot(1:length(S_col), S_col, 'o', 'LineWidth', 2)
set(gca,'Fontsize',12)
xlabel('Singular Values');
```

```
ylabel('Relative Energy');
title('Relative Energy of Singular Values - Test 4');

% plot of all camera angles showing major variation direction - z
subplot(2,2,1)
plot(1:length(y1_4),y1_4,1:length(y1_4),y2_4,1:length(y1_4),x3_4,'LineWidth',
1)
set(gca,'Xlim',[0,length(y1_4)],'Ylim',[-150,150],'Fontsize',12)
legend("Cam 1","Cam 2","Cam 3")
xlabel('Time (frames)');
ylabel('Displacement (pixels)');
title('Bucket Displacement in Time - Test 4');

% plot of data projections onto pricipal components
subplot(2,2,3)
A_proj = U'*A;
plot(1:length(A_proj(1,:)), A_proj(1,:),'LineWidth',1)
hold on
plot(1:length(A_proj(2,:)), A_proj(2,:),'LineWidth',1)
hold on
plot(1:length(A_proj(3,:)), A_proj(3,:),'LineWidth',1)
hold on
plot(1:length(A_proj(4,:)), A_proj(4,:),'LineWidth',1)
hold on
legend("P.C. 1","P.C. 2","P.C. 3","P.C. 4")
set(gca,'Xlim',[0,length(A_proj(1,:))],'Fontsize',12)
xlabel('Time (frames)');
ylabel('Displacement (pixels)');
title('Data Projections onto Principal Components - Test 4');
```