Principle Component Analysis on a Spring-Mass System

Yukai Yan

February 24, 2021

Abstract

In this paper, we try to illustrate various aspects of the Principle Component Analysis (PCA), its practical usefulness, and the effects of noise on the PCA algorithms. The study will be conducted on a spring-mass system, where the scenario was recorded by three cameras at different angles. Using PCA, we can understand the motion of the mass-spring system.

1 Introduction and Overview

1.1 A Spring-Mass System

Imagine we don't have much physics training, and we want to understand the mass-spring system. We experiment and capture the object's motion in a spring-mass system by three cameras at various angles. Combine all the recorded data, we now have a 6-dimensional dataset. Our goal is to extract the low-dimensional dynamics for analysis.

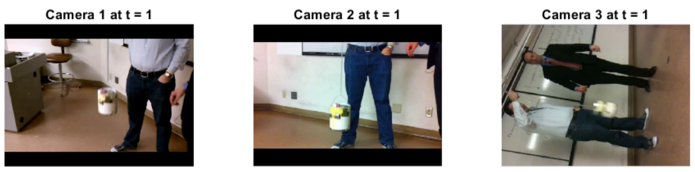

Figure 1: A snapshot of the spring-mass system by three cameras

1.2 Approach the problem

Working in high-dimensional spaces can be undesirable for many reasons. The raw data can record redundant information and possibly be sparse due to their dimensionality. Therefore, we want to transform data from a high-dimensional space to a low-dimensional space but retain their properties. This process is known as Dimensional Reduction in data analysis. The primary technique for dimensional reduction used in this paper is called principal component analysis. It will perform a linear mapping of the data to a lower-dimensional space s.t. the variables in the dataset are uncorrelated. Each variable contains completely new information that can be used for analysis.

1.3 Test cases on Principle Component Analysis

In an ideal world, the data can be obtained in a perfect scenario without noise. However, that is not always possible. Hence, we want to test PCA on various cases and analysis how it performs under extreme circumstances. In *test one(Ideal case)*, the entire motion is in the z direction with the simple harmonic motion being observed. In *test two(Noisy case)*, we repeat the idea case experiment, but the camera was shaking during recording, makes it harder to tract the simple harmonic motion. In *test three(Horizontal Displacement)*, the mass is released off-center, so it produces movement in the $x - y$ place as well as the z direction. In *test four(Horizontal Displacement and Rotation)*, the mass is released off-center, and it rotates so as to produce rotation in the $x - y$ place as well as the z direction.

1.4 Overview

We will discuss singular value decomposition, Principle Component Analysis, and data extraction in Section Two. In Section Three, we will develop the solution and implement the corresponding algorithms. Section Four will include the computational results derived from the approach, as the summary and conclusion will be in Section Five.

2 Theoretical Background

2.1 Principle Component Analysis

Suppose there are 10 assignments and each vector contains grades from the same 20 randomly chosen students, we can put each of these row vectors into a matrix and compute the covariance matrix between the rows of x with one matrix multiplication:

$$X = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}, \qquad C_X = \tfrac{1}{n-1} X X^T = \begin{bmatrix} \sigma_a^2 \sigma_{ab}^2 \sigma_{ac}^2 \sigma_{ad}^2 \\ \sigma_{ba}^2 \sigma_b^2 \sigma_{bc}^2 \sigma_{bd}^2 \\ \sigma_{ca}^2 \sigma_{cb}^2 \sigma_c^2 \sigma_{cd}^2 \\ \sigma_{da}^2 \sigma_{db}^2 \sigma_{dc}^2 \sigma_d^2 \end{bmatrix}$$

where the covariance can be computed by the formula:

$$\sigma_a^2 = \frac{1}{n-1} a a^T, \qquad \sigma_{ab}^2 = \frac{1}{n-1} a b^T$$

The goal of principle component analysis is to find a new set of coordinates (a change of basis) so that the variables are now uncorrelated. It would be nice to know which variable have the largest variance because these contain most important information about our data. Therefore, we want to diagonalize this matrix so that all off-diagonal elements (covariances) are zero:

$$C_X = V \Lambda V^{-1}$$

Note that The basis of eigenvectors contained in V are called principle components, they are uncorrelated since they are orthogonal.

2.2 Singular Value Decomposition

In matrix form, the singular value decomposition of the matrix A with size $m \times n$ can be represented with:

$$A = U \Sigma V^*$$

Where U is an $m \times m$ complex unitary matrix, Σ is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and V is an $n \times n$ complex unitary matrix. The singular value decomposition of matrix A is connected to the eigenvalue decomposition of AA^T. To account for the $\frac{1}{n-1}$ factor, consider:

$$A = \frac{1}{\sqrt{n-1}} X$$

Then, for principle component analysis, we have:

$$C_X = \frac{1}{n-1} X X^T = A A^T = U \Sigma^2 U^T$$

So, the eigenvalues of the covariance matrix are the squared of the scaled singular values. Since we used a change of basis to work in the basis of the principal components, the data in the new coordinate and its covariance can be obtained by:

$$Y = U^T X, \qquad C_Y = \Sigma^2$$

2.3 Data extraction

To perform the PCA, we need to extract the location of the bucket from the videos. In this project, we crop the area of interest and track the bucket by identifying the white spot. We will introduce the implementation of tracking the bucket's location in section 3, and a snapshot of the filtered video is shown in section 4.

3 Algorithm Implementation and Development

Since our goal is to analysis how PCA performs under various cases mentioned in section 1, we will apply PCA to the video data. The code can be into four main parts:

1. Load data and the bucket's location.
2. Data extraction algorithm
3. Apply the singular value decomposition
4. Apply the Principle Component Analysis

Note that the code implemented in MatLab is included in Appendix B.

```
Algorithm 1: Load Data and the bucket's location
  Clean work space
  Load video data from 3 cameras for appropriate tests.
  Extract $x, y$ coordinates from the data for each camera.
  Align data by subtracting the mean for each column.
```

Extracting x, y coordinates is done by a helper function, which is defined in the next algorithm.

```
Algorithm 2: Data extraction algorithm
  for each frame in the video do
      Find the $r, g, b$ components of that frame.
      Define the binary image from the frame where each pixel's $r, g, b$ is greater than certain threshold.
      Crop the area of interest and discard the remaining.
      Remove tiny spots outside the bucket (noise)
      Extract $x, y$ coordinates from the binary image.
  end
```

The bucket area will label as white and everything else is black in the binary image. The x, y coordinates can be obtained by finding the white area. An example of the filtered image is shown in section 4.

```
Algorithm 3: Apply the singular value decomposition
  Define $X$ to be the matrix of combined $x, y$ coordinates.
  Find $U, S, V$ by performing the singular value decomposition on $X$.
  Define $\Sigma$ to be the diagonal of the matrix $S$.
  Plot $\Sigma$, energy and cumulative energy.
```

With U, S, V, we can perform the principle component analysis in the next step.

```
Algorithm 4: Apply the Principle Component Analysis
  Define $Y$ to be the matrix multiplication of $U^T$ and $X$
  Plot modes and the principle component $Y$.
```

The results of the computed data (along with the graph) are included in Section 4.

4 Computational Results

For reference, the bucket's location can be obtained by the following filtered frames at any give time:

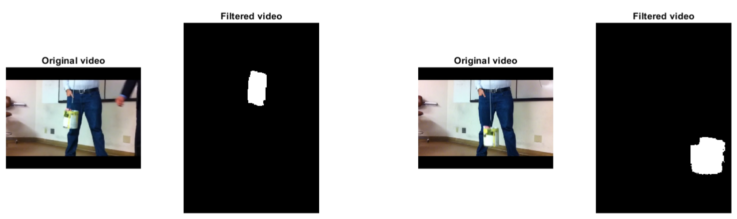

Figure 2: Snapshots of the original and filtered video

With the bucket Location, we can now perform the principle component analysis.

4.1 Test 1: Ideal Case

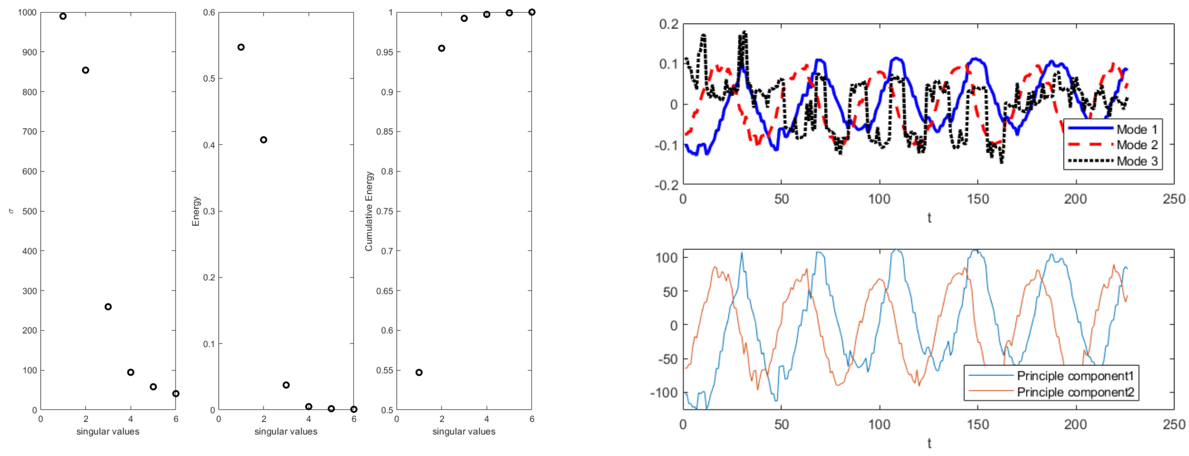

Figure 3: The singular values and the principle components for test 1

In an ideal case, not much noise was introduced, and the bucket's movement can be clearly identified in the vertical direction. From the graph, we can conclude that the first two singular values contain more than 95% of energies, which implies that the dominant modes capture the vertical movement. Note mode 3 introduces some errors to the data. This is caused by minor camera shaking in the third video. However, We are still able to estimate the bucket's location by the principal components Analysis.

4.2 Test 2: Noisy Case

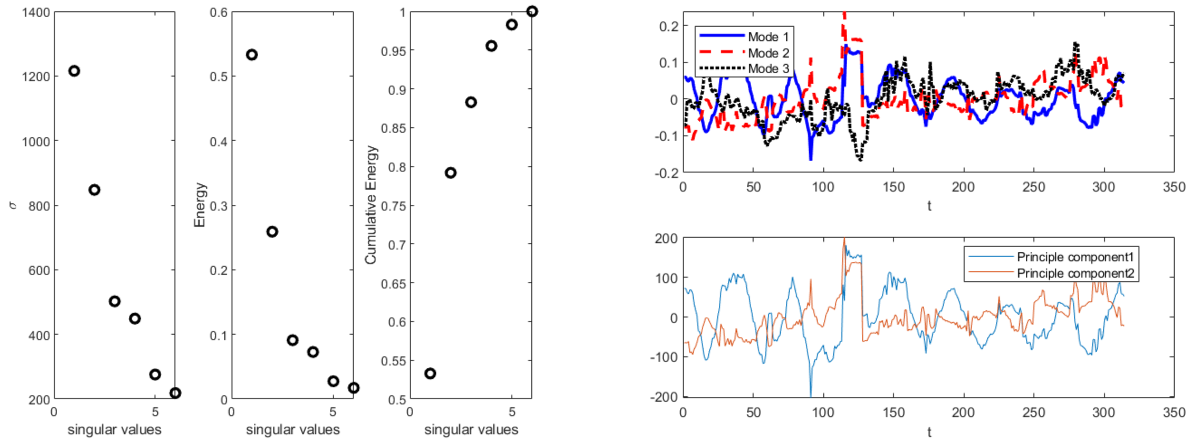

Figure 4: The singular values and the principle components for test 2

Unlike the previous test case, the camera shakes significantly during the video recording, introduce a lot of noise in our data. As you can tell from the graph, it requires four singular values to capture more than 95% of data. It is also hard to predict how these modes evolve in time. The amount of shaking creates difficulty for extracting the bucket's location; despite best efforts, we cannot reproduce the same bucket's location. Therefore, principal components Analysis is not practical in this test case.

4.3 Test 3: Horizontal Displacement

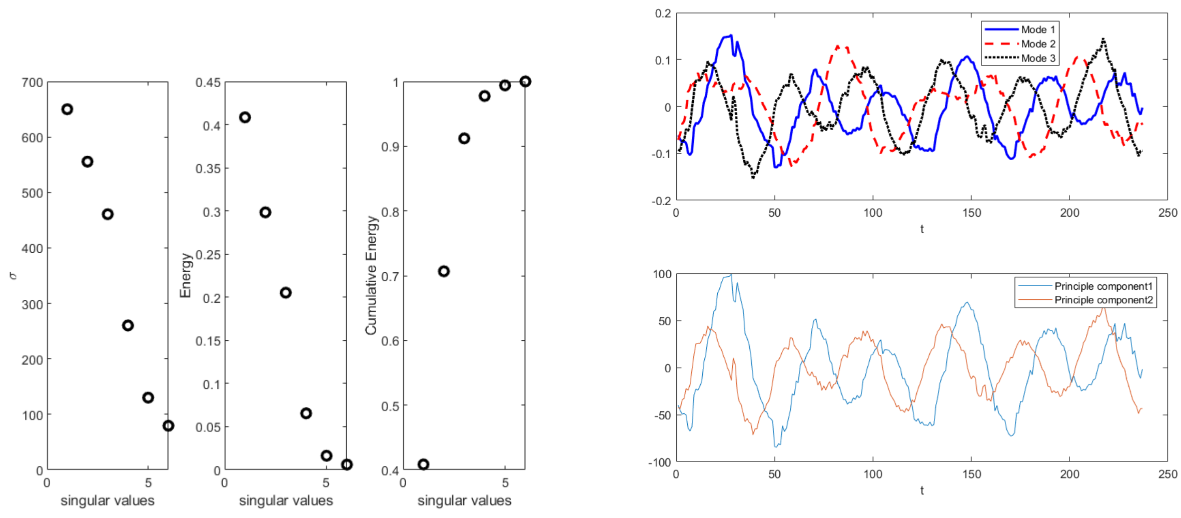

Figure 5: The singular values and the principle components for test 3

Unlike test two, only some smaller noise was introduced, and the bucket was released off-center to produce motion in the $x - y$ plane as well as the z direction. Like test two, it requires four significant values to capture more than 95% of the data, but a reasonable estimation of the bucket's position can be obtained by the dominant modes. Although they are not robust as what's showed in test 1, as the bucket moves in a bigger area, it is still a reasonable estimate of its location.

4.4 Test 4: Horizontal Displacement and Rotation

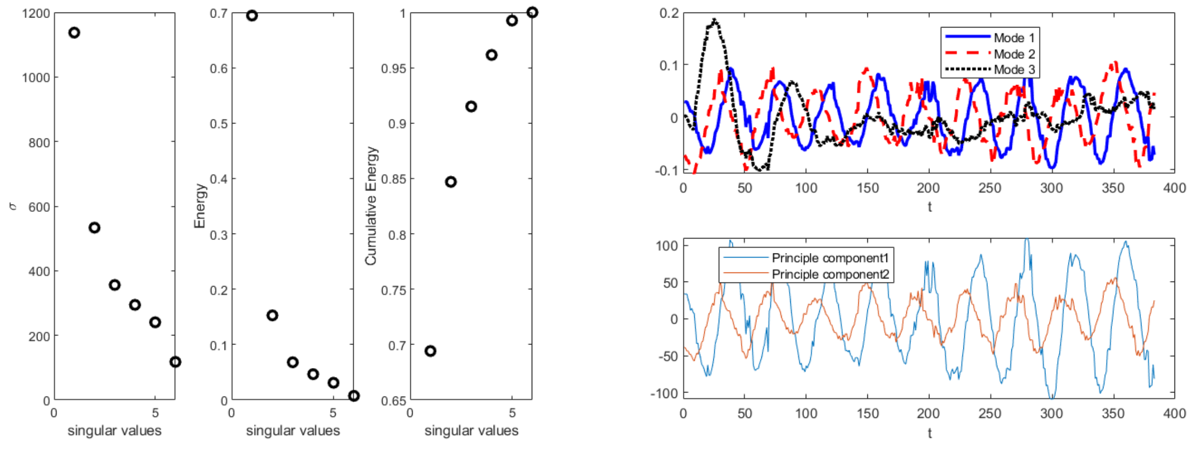

Figure 6: The singular values and the principle components for test 4

In this test, the bucket is released off-center and rotates to produce rotation in the $x - y$ plane and the motion in the z direction. Like previous tests, it requires four significant values to capture more than 95% of the data, but a robust estimation of the bucket's position can be obtained by the dominant modes. The rotation does not seem to have any negative consequences on the analysis. Interestingly, the rotation makes the bucket easier to attract, producing a robust estimation like the test one.

5 Summary and Conclusions

The principal component analysis allows us to transform data from a high-dimensional space to a low-dimensional space and retain their properties. This paper demonstrates that we can reduce multidimensional data recorded by the cameras into 2-dimensional data for analysis. It is capable of generating a reasonable estimate of the bucket's movement. However, not all data obtained can be clean and free of noise. When we introduced a significant amount of noise to the data, the bucket's location becomes hard to tract, and PCA does little on noise filtering. In summary, principal component analysis is a powerful tool for dimensionality reduction, but its shortcomings are its ability to clean noise. Future work can be done to design a machine learning model to cancel noises and study the noise's impact on PCA.

Appendices

Appendix A: MatLab functions used and brief explanations

The following list includes the MatLab functions used in Appendix B. It also contains a brief implementation explanation of their functionalities. More information can be found at MatLab Documentation at MathWorks.

- *load(filename)*: loads data from filename.
- *implay(filename)*: opens the Video Viewer app, displaying the content of the file specified by *filename*.
- *szdim = size(A,dim)*: returns the length of dimension *dim* when *dim* is a positive integer scalar.
- *Icropped = imcrop(I,rect)*: crops the image *I* according to the position and dimensions specified in the crop rectangle *rect*
- *BW2 = bwareaopen(BW,P)*: removes all connected components (objects) that have fewer than *P* pixels from the binary image *BW*, producing another binary image, *BW2*.
- *k = find(X)*: returns a vector with the same orientation as *X*.
- *[row,col] = ind2sub(sz,ind)*: returns the arrays row and col containing the equivalent row and column subscripts corresponding to the linear indices ind for a matrix of size *sz*.
- *imshow(I)*: displays the grayscale image *I* in a figure.
- *M = mean(A)*: returns the mean of the elements of *A* along the first array dimension whose size does not equal 1
- *M = min(A)*: returns the minimum elements of an array.
- *[U,S,V] = svd(A,'econ')*: produces an economy-size decomposition of m-by-n matrix *A*
- *D = diag(v)*: returns a square diagonal matrix with the elements of vector *v* on the main diagonal.

Appendix B: MatLab Code

Note that one MatLab file was created for each test, and the included files were in order with the tests. The MatLab files were almost the same, with slight variations in Algorithm1: load data and Algorithm2: Data extraction.

Test 1 MatLab Code

```matlab
%% load data for Test1: idea case
clear variables; close all; clc;

load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')

% display video
% implay(vidFrames1_1)

% extract the location of the bucket for test1
threshold = 220;
[xpos1, ypos1] = extractLocation(vidFrames1_1, threshold, 1, false);
[xpos2, ypos2] = extractLocation(vidFrames2_1, threshold, 2, false);
[ypos3, xpos3] = extractLocation(vidFrames3_1, threshold, 3, false);

% clean and align data for graphing
minSize = min(length(xpos1), min(length(xpos2), length(xpos3)));
xpos1 = xpos1(1:minSize); xpos1 = xpos1 - mean(xpos1);
ypos1 = ypos1(1:minSize); ypos1 = ypos1 - mean(ypos1);
xpos2 = xpos2(1:minSize); xpos2 = xpos2 - mean(xpos2);
ypos2 = ypos2(1:minSize); ypos2 = ypos2 - mean(ypos2);
xpos3 = xpos3(1:minSize); xpos3 = xpos3 - mean(xpos3);
ypos3 = ypos3(1:minSize); ypos3 = ypos3 - mean(ypos3);

%% Singular value decomposition
X = [xpos1';ypos1';xpos2';ypos2';xpos3';ypos3'];
[U,S,V] = svd(X,'econ');

% Determine energy from the full system contained in each mode
sig = diag(S);
figure(2)
subplot(1,3,1); plot(sig,'ko','Linewidth',2);
ylabel('\sigma');xlabel('singular values')
subplot(1,3,2); plot(sig.^2/sum(sig.^2),'ko','Linewidth',2);
ylabel('Energy');xlabel('singular values')
subplot(1,3,3);plot(cumsum(sig.^2)/sum(sig.^2),'ko','Linewidth',2)
ylabel('Cumulative Energy');xlabel('singular values')

%% Principle Component Analysis
t = 1:minSize;
figure(3)
subplot(2,1,1)
plot(t,V(:,1),'b',t,V(:,2),'--r',t,V(:,3),':k','Linewidth',2)
legend('Mode 1','Mode 2','Mode 3', 'Location','best');
xlabel('t')

subplot(2,1,2)
X_proj = U' * X;
plot(t, X_proj(1,:), t, X_proj(2,:));
legend('Principle component1','Principle component2', 'Location','best');
xlabel('t')

```

```matlab

%% plot original graph and rank-1 rank-2 rank3 approximations
% not used in the paper, used to check bucket positions
t = 1:minSize;
figure(5)
subplot(2,2,1)
plot(t,xpos1,t,ypos1,t,xpos2,t,ypos2,t,xpos3,t,ypos3);
legend('x1','y1','x2','y2','x3','y3')
ylabel('Distance from mean'); xlabel('time');title('Original');
for j=1:3
    ff = U(:,1:j)*S(1:j,1:j)*V(:,1:j)';
    subplot(2,2,j+1)
    plot(t,ff)
    legend('x1','y1','x2','y2','x3','y3')
    ylabel('Distance from mean'); xlabel('time'); title(['rank ', num2str(j)]);
end


%% helper funcitons
% extract the location of the bucket by identifying the white parts
% @Input  VidFrame: video data
%         threshold: used to identify the white color
%         camera: the camera number
%         showGraph: plot comparison graph in the process
% @Output [xpos ypos]: the coordinates of the bucket
% see in-code comments for details
function [xpos, ypos] = extractLocation(vidFrame, threshold, camera, showGraph)
numFrames = size(vidFrame,4);
xpos = [];
ypos = [];

for i = 1:numFrames
    frame = vidFrame(:,:,:,i);
    r = frame(:,:,1);
    g = frame(:,:,2);
    b = frame(:,:,3);

    % Find white, where each color channel is more than threshold
    binaryImage = r > threshold & g > threshold & b > threshold;

    % crop image, data obtained using imcrop
    if (camera == 1)
        binaryImage = imcrop(binaryImage,[292.5 81.5 112 335]);
    elseif (camera == 2)
        binaryImage = imcrop(binaryImage,[228.5 82.5 141 332]);
        % remove extra white space located on the topleft
        for x = 1:80
            for y = 1:80
                binaryImage(x,y) = 0;
            end
        end
    elseif (camera == 3)
        binaryImage = imcrop(binaryImage,[171.5 238.5 394 125]);
    end

    % Get rid of blobs smaller than 50 pixels.
    binaryImage = bwareaopen(binaryImage, 50);

    [x,y] = ind2sub(size(binaryImage),find(binaryImage,1));
    xpos = [xpos; x];
    ypos = [ypos; y];

    % show cleaned graph
    if (showGraph)
        figure(1)
        subplot(1,2,1),imshow(frame); title('Original video');
        subplot(1,2,2),imshow(binaryImage); title('Filtered video');
        drawnow
    end
end

end
```

Test 2 MatLab Code

```
%% load data for Test2: Noisy case
clear variables; close all; clc;

load('cam1_2.mat')
load('cam2_2.mat')
load('cam3_2.mat')

% display video
% implay(vidFrames1_2)

% extract the location of the bucket for test2
threshold = 220;
[xpos1, ypos1] = extractLocation(vidFrames1_2, threshold, 1, false);
[xpos2, ypos2] = extractLocation(vidFrames2_2, threshold, 2, false);
[ypos3, xpos3] = extractLocation(vidFrames3_2, threshold, 3, false);

% clean and align data for graphing
minSize = min(length(xpos1), min(length(xpos2), length(xpos3)));
xpos1 = xpos1(1:minSize); xpos1 = xpos1 - mean(xpos1);
ypos1 = ypos1(1:minSize); ypos1 = ypos1 - mean(ypos1);
xpos2 = xpos2(1:minSize); xpos2 = xpos2 - mean(xpos2);
ypos2 = ypos2(1:minSize); ypos2 = ypos2 - mean(ypos2);
xpos3 = xpos3(1:minSize); xpos3 = xpos3 - mean(xpos3);
ypos3 = ypos3(1:minSize); ypos3 = ypos3 - mean(ypos3);

%% Singular value decomposition
X = [xpos1';ypos1';xpos2';ypos2';xpos3';ypos3'];
[U,S,V] = svd(X,'econ');

% Determine energy from the full system contained in each mode
sig = diag(S);
figure(2)
subplot(1,3,1); plot(sig,'ko','Linewidth',2);
ylabel('\sigma');xlabel('singular values')
subplot(1,3,2); plot(sig.^2/sum(sig.^2),'ko','Linewidth',2);
ylabel('Energy');xlabel('singular values')
subplot(1,3,3);plot(cumsum(sig.^2)/sum(sig.^2),'ko','Linewidth',2)
ylabel('Cumulative Energy');xlabel('singular values')

%% Principle Component Analysis
t = 1:minSize;
figure(3)
subplot(2,1,1)
plot(t,V(:,1),'b',t,V(:,2),'--r',t,V(:,3),':k','Linewidth',2)
legend('Mode 1','Mode 2','Mode 3', 'Location','best');
xlabel('t')

```

```
subplot(2,1,2)
X_proj = U' * X;
plot(t, X_proj(1,:), t, X_proj(2,:));
legend('Principle component1','Principle component2', 'Location','best');
xlabel('t')


%% plot original graph and rank-1 rank-2 rank3 approximations
% not used in the paper, used to check bucket positions
t = 1:minSize;
figure(5)
subplot(2,2,1)
plot(t,xpos1,t,ypos1,t,xpos2,t,ypos2,t,xpos3,t,ypos3);
legend('x1','y1','x2','y2','x3','y3')
ylabel('Distance from mean'); xlabel('time');title('Original');
for j=1:3
    ff = U(:,1:j)*S(1:j,1:j)*V(:,1:j)';
    subplot(2,2,j+1)
    plot(t,ff)
    legend('x1','y1','x2','y2','x3','y3')
    ylabel('Distance from mean'); xlabel('time'); title(['rank ', num2str(j)]);
end


%% helper funcitons
% extract the location of the bucket by identifying the white parts
% @Input  VidFrame: video data
%         threshold: used to identify the white color
%         camera: the camera number
%         showGraph: plot comparison graph in the process
% @Output [xpos ypos]: the coordinates of the bucket
% see in-code comments for details
function [xpos, ypos] = extractLocation(vidFrame, threshold, camera, showGraph)
numFrames = size(vidFrame,4);
xpos = [];
ypos = [];

for i = 1:numFrames
    frame = vidFrame(:,:,:,i);
    r = frame(:,:,1);
    g = frame(:,:,2);
    b = frame(:,:,3);

    % Find white, where each color channel is more than threshold
    binaryImage = r > threshold & g > threshold & b > threshold;

    % crop image, data obtained using imcrop
    if (camera == 1)
        binaryImage = imcrop(binaryImage,[292.5 81.5 112 335]);
        % remove extra white space located on the topleft
        for x = 1:80
            for y = 1:120
                binaryImage(x,y) = 0;
            end
        end
    elseif (camera == 2)
        binaryImage = imcrop(binaryImage,[222.5 64.5 200 361]);
        % remove extra white space located on the topleft
        for x = 1:100
            for y = 1:100
                binaryImage(x,y) = 0;
            end
        end
    elseif (camera == 3)
        binaryImage = imcrop(binaryImage,[206.5 132.5 400 268]);
    end

    % Get rid of blobs smaller than 100 pixels.
    binaryImage = bwareaopen(binaryImage, 100);

    [x,y] = ind2sub(size(binaryImage),find(binaryImage,1));
    xpos = [xpos; x];
    ypos = [ypos; y];

    % show cleaned graph
    if (showGraph)
        figure(1)
        subplot(1,2,1),imshow(frame); title('Original video');
        subplot(1,2,2),imshow(binaryImage); title('Filtered video');
        drawnow
    end
end

end
```

Test 3 MatLab Code

```
%% load data for Test3: Horizontal Displacement
clear variables; close all; clc;

load('cam1_3.mat')
load('cam2_3.mat')
load('cam3_3.mat')

% display video
% implay(vidFrames2_3)

% extract the location of the bucket for test3
threshold = 220;
[xpos1, ypos1] = extractLocation(vidFrames1_3, threshold, 1, false);
[xpos2, ypos2] = extractLocation(vidFrames2_3, threshold, 2, false);
[ypos3, xpos3] = extractLocation(vidFrames3_3, threshold, 3, false);

% clean and align data for graphing
minSize = min(length(xpos1), min(length(xpos2), length(xpos3)));
xpos1 = xpos1(1:minSize); xpos1 = xpos1 - mean(xpos1);
ypos1 = ypos1(1:minSize); ypos1 = ypos1 - mean(ypos1);
xpos2 = xpos2(1:minSize); xpos2 = xpos2 - mean(xpos2);
ypos2 = ypos2(1:minSize); ypos2 = ypos2 - mean(ypos2);
xpos3 = xpos3(1:minSize); xpos3 = xpos3 - mean(xpos3);
ypos3 = ypos3(1:minSize); ypos3 = ypos3 - mean(ypos3);

%% Singular value decomposition
X = [xpos1';ypos1';xpos2';ypos2';xpos3';ypos3'];
[U,S,V] = svd(X,'econ');

% Determine energy from the full system contained in each mode
sig = diag(S);
figure(2)
subplot(1,3,1); plot(sig,'ko','Linewidth',2);
ylabel('\sigma');xlabel('singular values')
subplot(1,3,2); plot(sig.^2/sum(sig.^2),'ko','Linewidth',2);
```

```
ylabel('Energy');xlabel('singular values')
subplot(1,3,3);plot(cumsum(sig.^2)/sum(sig.^2),'ko','Linewidth',2)
ylabel('Cumulative Energy');xlabel('singular values')

%% Principle Component Analysis
t = 1:minSize;
figure(3)
subplot(2,1,1)
plot(t,V(:,1),'b',t,V(:,2),'--r',t,V(:,3),':k','Linewidth',2)
legend('Mode 1','Mode 2','Mode 3', 'Location','best');
xlabel('t')

subplot(2,1,2)
X_proj = U' * X;
plot(t, X_proj(1,:), t, X_proj(3,:));
legend('Principle component1','Principle component2', 'Location','best');
xlabel('t')

%% plot original graph and rank-1 rank-2 rank3 approximations
% not used in the paper, used to check bucket positions
t = 1:minSize;
figure(5)
subplot(2,2,1)
plot(t,xpos1,t,ypos1,t,xpos2,t,ypos2,t,xpos3,t,ypos3);
legend('x1','y1','x2','y2','x3','y3')
ylabel('Distance from mean'); xlabel('time');title('Original');
for j=1:3
    ff = U(:,1:j)*S(1:j,1:j)*V(:,1:j)';
    subplot(2,2,j+1)
    plot(t,ff)
    legend('x1','y1','x2','y2','x3','y3')
    ylabel('Distance from mean'); xlabel('time'); title(['rank ', num2str(j)]);
end


%% helper funcitons
% extract the location of the bucket by identifying the white parts
% @Input  VidFrame: video data
%         threshold: used to identify the white color
%         camera: the camera number
%         showGraph: plot comparison graph in the process
% @Output [xpos ypos]: the coordinates of the bucket
% see in-code comments for details
function [xpos, ypos] = extractLocation(vidFrame, threshold, camera, showGraph)
numFrames = size(vidFrame,4);
xpos = [];
ypos = [];

%for i = 6:24
for i = 1:numFrames
    frame = vidFrame(:,:,:,i);
    r = frame(:,:,1);
    g = frame(:,:,2);
    b = frame(:,:,3);

    % Find white, where each color channel is more than threshold
    binaryImage = r > threshold & g > threshold & b > threshold;

    % crop image, data obtained using imcrop
    if (camera == 1)
        binaryImage = imcrop(binaryImage,[260.5 143.5 171 288]);
    elseif (camera == 2)
        binaryImage = imcrop(binaryImage,[188.5 174.5 175 246]);
        if (i >= 6 && i <= 24)
            for x = 1:200
                for y = 1:100
                    binaryImage(x,y) = 0;
                end
            end
        end

    elseif (camera == 3)
        binaryImage = imcrop(binaryImage,[181.5 133.5 448 252]);
    end

    % Get rid of blobs smaller than 200 pixels.
    binaryImage = bwareaopen(binaryImage, 200);

    [x,y] = ind2sub(size(binaryImage),find(binaryImage,1));
    xpos = [xpos; x];
    ypos = [ypos; y];

    % show cleaned graph
    if (showGraph)
        figure(1)
        subplot(1,2,1),imshow(frame); title('Original video');
        subplot(1,2,2),imshow(binaryImage); title('Filtered video');
        drawnow
    end
end

end
```

Test 4 MatLab Code

```
%% load data for Test4: Horizontal Displacement + rotation
clear variables; close all; clc;

load('cam1_4.mat')
load('cam2_4.mat')
load('cam3_4.mat')

% display video
% implay(vidFrames1_4)

% extract the location of the bucket for test4
threshold = 220;
[xpos1, ypos1] = extractLocation(vidFrames1_4, threshold, 1, false);
[xpos2, ypos2] = extractLocation(vidFrames2_4, threshold, 2, false);
[ypos3, xpos3] = extractLocation(vidFrames3_4, threshold, 3, false);

% clean and align data for graphing
minSize = min(length(xpos1), min(length(xpos2), length(xpos3)));
xpos1 = xpos1(1:minSize); xpos1 = xpos1 - mean(xpos1);
ypos1 = ypos1(1:minSize); ypos1 = ypos1 - mean(ypos1);
xpos2 = xpos2(1:minSize); xpos2 = xpos2 - mean(xpos2);
ypos2 = ypos2(1:minSize); ypos2 = ypos2 - mean(ypos2);
xpos3 = xpos3(1:minSize); xpos3 = xpos3 - mean(xpos3);
ypos3 = ypos3(1:minSize); ypos3 = ypos3 - mean(ypos3);

%% Singular value decomposition
X = [xpos1';ypos1';xpos2';ypos2';xpos3';ypos3'];
```

```
[U,S,V] = svd(X,'econ');

% Determine energy from the full system contained in each mode
sig = diag(S);
figure(2)
subplot(1,3,1); plot(sig,'ko','Linewidth',2);
ylabel('\sigma');xlabel('singular values')
subplot(1,3,2); plot(sig.^2/sum(sig.^2),'ko','Linewidth',2);
ylabel('Energy');xlabel('singular values')
subplot(1,3,3);plot(cumsum(sig.^2)/sum(sig.^2),'ko','Linewidth',2)
ylabel('Cumulative Energy');xlabel('singular values')

%% Principle Component Analysis
t = 1:minSize;
figure(3)
subplot(2,1,1)
plot(t,V(:,1),'b',t,V(:,2),'--r',t,V(:,3),':k','Linewidth',2)
legend('Mode 1','Mode 2','Mode 3', 'Location','best');
xlabel('t')

subplot(2,1,2)
X_proj = U' * X;
plot(t, X_proj(1,:), t, X_proj(2,:));
legend('Principle component1','Principle component2', 'Location','best');
xlabel('t')


%% plot original graph and rank-1 rank-2 rank3 approximations
% not used in the paper, used to check bucket positions
t = 1:minSize;
figure(5)
subplot(2,2,1)
plot(t,xpos1,t,ypos1,t,xpos2,t,ypos2,t,xpos3,t,ypos3);
legend('x1','y1','x2','y2','x3','y3')
ylabel('Distance from mean'); xlabel('time');title('Original');
for j=1:3
    ff = U(:,1:j)*S(1:j,1:j)*V(:,1:j)';
    subplot(2,2,j+1)
    plot(t,ff)
    legend('x1','y1','x2','y2','x3','y3')
    ylabel('Distance from mean'); xlabel('time'); title(['rank ', num2str(j)]);
end


%% helper funcitons
% extract the location of the bucket by identifying the white parts
% @Input  VidFrame: video data
%         threshold: used to identify the white color
%         camera: the camera number
%         showGraph: plot comparison graph in the process
% @Output [xpos ypos]: the coordinates of the bucket
% see in-code comments for details
function [xpos, ypos] = extractLocation(vidFrame, threshold, camera, showGraph)
numFrames = size(vidFrame,4);
xpos = [];
ypos = [];

for i = 1:numFrames
    frame = vidFrame(:,:,:,i);
    r = frame(:,:,1);
    g = frame(:,:,2);
    b = frame(:,:,3);

    % Find white, where each color channel is more than threshold
    binaryImage = r > threshold & g > threshold & b > threshold;

    % crop image, data obtained using imcrop
    if (camera == 1)
        binaryImage = imcrop(binaryImage,[260.5 143.5 171 288]);
    elseif (camera == 2)
        binaryImage = imcrop(binaryImage,[228.5 82.5 141 332]);
        % remove extra white space located on the topleft
        for x = 1:300
            for y = 1:30
                binaryImage(x,y) = 0;
            end
        end
    elseif (camera == 3)
        binaryImage = imcrop(binaryImage,[181.5 133.5 448 252]);
    end

    % Get rid of blobs smaller than 100 pixels.
    binaryImage = bwareaopen(binaryImage, 100);

    [x,y] = ind2sub(size(binaryImage),find(binaryImage,1));
    xpos = [xpos; x];
    ypos = [ypos; y];

    % show cleaned graph
    if (showGraph)
        figure(1)
        subplot(1,2,1),imshow(frame); title('Original video');
        subplot(1,2,2),imshow(binaryImage); title('Filtered video');
        drawnow
    end
end

end
```