Homework 3: PCA Analysis: Robustness to Noise and Unwanted Deviation for Harmonic Oscillations

Hongda Li

February 23, 2021

Abstract

We are interested in bench marking the efficacy and behavior of the Principal Component method by applying it to a physical setup up. Three cameras were placed in different orientation to observe a can moving in harmonic motion. There are several cases to compare with. Noise in the form of camera vibration were introduced and non-linearity of the motions where introduced by mixing the harmonic motion with some degree of pendulum oscillations and rotations for some of the cases.

1 Introduction and Overview

Our objective is to analyze harmonic motion captured by cameras in different orientations using the idea of Principal Component to reduce the dimensions of the observed data. Along with this objective, we are interested in potential impact on the PCA algorithm if noise and unwanted deviation of the motion were included.

The physical setup consists of 3 camera, placed in different position and orientation, each has a recording of a can in harmonic motion along some axis on the screen of the camera. Four cases were included, each has different level of noise and degree of deviation from the ideal harmonic motion. The first case establish ideal behavior of the can motion, creating a baseline for comparing the results obtained from the PCA method.

1. Ideal Case: Three cameras placed in different position, observing a can moving in simple harmonic motion along one axis.
2. Noisy Case: Three cameras placed in different position, in addition to the first case, noisy in the form of camera vibrations are introduced.
3. Horizontal Displacement: In addition to the first case, the mass is released off center, causing the can to have pendulum, orthogonal to the direction of the harmonic motion.
4. Horizontal Displacement with Rotation: In addition to case 3, The can starts to rotates as it undergoes harmonic and pendulum motions.

The major challenges are the pre-processing the data, interpreting and comparing the Principal Components. A solid understanding of the theory and the setup of the problems are needed.

2 Theoretical Background

2.1 Principal Value Decomposition

The idea is to use Singular Value Decomposition (SVD). The singular value decomposition of a $m \times n$ can be considered to be:

$$A = U\Sigma V^T = \sum_{j=1}^{\min(m,n)} \left(u_j \sigma_j v_j^T\right) \tag{1}$$

The U, V are Orthogonal Matrices and Σ is the diagonal matrix. For our application, we only consider real matrices so discussion will be using Orthogonal instead of unitary, and transpose instead of Hermitian Adjoint.

And for our sake, we consider the economic decomposition of the singular value, meaning that Σ is a squared matrix with dimension $k = \min(m, n)$, U matrix is $m \times k$ and V matrix is $n \times k$. SVD represents the matrix in term of 1 rotation in the row space, a stretching transformation and another rotation in the column space of matrix. The idea of Principal Value Analysis (PCA) uses SVD.

2.2 PCA Using SVD

The physical set up of the problem allows us to represent the motions of the can in 2D using points (x_i, y_i) for each frame captured by each camera. There are 3 cameras in total, and therefore, we will be able to capture 6 stream of data. Packing them into a matrix where each row contain coordinate data captured by one of the camera we have:

$$X = \begin{bmatrix} x_a^T \\ y_a^T \\ x_b^T \\ y_b^T \\ x_c^T \\ y_c^T \end{bmatrix} \tag{2}$$

Where each 2 rows are the x, y coordinates of the can in the frame of 3 cameras indexed by a, b, c.

The key idea here is that, if all of these observed data were describing one single harmonic motions (which is not the case but let's assume that), then there exists a unitary transformation (An alternative orthogonal basis) of the data such that x only has one degree of freedom, equivalent to it being low rank[1].

Consider the co-variance matrix and our goal is to show that the SVD can be used to extract an alternative basis for the motions such that, it gets rid of the non-diagonal entries of the co-variance matrix when the motion is described under the new basis. And we also get the Principal Components for the motion during this process. Consider the Co-variance matrix for the row data matrix:

$$\text{Cov(X)} = \frac{1}{n-1} X X^T \tag{3}$$

If x_a, y_a were describing the same motion, then the co-variance between the 2 vectors, x_a^T, y_a^T will far away from zero. The highest co-variance of each row is on the diagonal of the matrix C_x due to the fact that the diagonal is the variance for each row of the data matrix. Applying SVD on the row data matrix X, we have: $X = U\Sigma V^T$, the alternative basis here will be the U matrix. **Note that**: the mean of each row of the data matrix has to be zero in order for co-variance matrix to actually make sense.

The proof can shown by considering the argument:

$$\begin{aligned} \text{Cov}(U^T X)(n-1) &= U^T X (U^T X)^T \\ \text{Cov}(U^T X)(n-1) &= U^T X X^T U \\ \text{Cov}(U^T X)(n-1) &= U^T U \Sigma V^T V \Sigma U^T U \\ \text{Cov}(U^T X)(n-1) &= \Sigma^2 \end{aligned} \tag{4}$$

Observe that, by projecting onto the U^T matrix, the transformed data matrix is diagonal, indicating minimal non-diagonal elements. Diagonal Co-variance Matrix implies that each row of the new data matrix is observing a unique composition of the signals of the original data entries. Intuitively, it could means a motion that is not correlated to any other motions observed. The motions under alternative basis will be: ΣV^T accordingly.

For our physical set up, the matrix U is representing an optimal mixing (A different perspective) of the x, y motions captured by each cameras. And the matrix V will be the actual principal motions (in 1D) that the matrix U is trying to mix, and the matrix Σ will scale the signal coming out from V^T. And this is my intuitive understanding of the matter.

3 Algorithm Implementation and Development

The same procedures were repeated for 4 of the case. It can be summarized by the following:

1. Motions extraction from frames of the video captured by each cameras.
2. Visualizing and standardizing the data.
3. Visualizing the principal components and distribution of the principal values.

The challenging part of the algorithm is motion extraction from the data, and the subtle part is the normalization of the data matrix. Both plays important roles for the analysis of the PCA algorithms.

3.1 Motion Extraction

Algorithm 1: Item Locator

```
Input: Background: M, Item: T; StartingPosition: X, Y; SearchRange [RangeX, RangeY]
Displacement:=[0, 0]; MinDifference = +inf
for II = -RangeX: RangeX - 1 do
  for JJ = -RangeY: RangeY - 1 do
    Xstart := X + II; Ystart = Y + JJ
    Xend := Xstart size(T, 1); Yend = Ystart + size(T, 2)
    if Any of Xstart, Ystart, Xend, Yend is Out of Background then
      continue
    end if
    Difference := abs(T - Background(Xstart: Xend, Ystart: Yend))
    if Difference < MinDifference then
      Displacement = [II; JJ]
      MinDifference := Mindiff
    end if
  end for
end for
Ouput: Displacement
```

Given an gray scale image, Algorithm 1 will search over a range on a gray scale background, minimizing the one norm of the difference. The item is a matrix, denoted as "T", and the background is another matrix. The top right corner of "T" is anchored at the starting position, and it will then slide around with "RangeX, RangeY" to minimize the 1-Norm difference between the "T" and the background.

For the first frame, the image of the can and the coordinate is selected manually. For each frame, we then Algorithm 1 to search for the item, and then we update the image for the can. Repeat the process for all remaining frames. **Note**: The parameter "X, Y" describes the indices of matrix representing each frame of the video, hence "X" is the vertical displacement of the can on screen and "Y" is the horizontal displacement of the can on the screen. The algorithm is implemented in FrameSearch.m

Algorithm 2: Route Tracer

```
Input: Video: V, StartingPosition: [X0, Y0], Item: T
Route := StartingPosition
for Frame F in V do
  Displacement := Use Algorithm 1 with T, StartingPosition: X0, Y0, and SearchRange: [30, 30]
  Route(:, end + 1) := Route(:, end) + Displacement
  Update: T using Displacement
  StartingPosition := StartingPosition + Displacement
end for
Return: Route
```

Algorithm 2 uses the first algorithm as a subroutine to find the displacement of the can for each frame, accumulating it into "Route".

This is used to extract the motion of the can for all 3 cameras for all 4 cases. At the start of the each video, the position of the can and a window that contains the can is identified manually. All the initial position of the can for each video is configured by TracerSettings.m and the algoritm that locates the can's location in pixels for each video is implemented by the "traceRoute" function in Tracer.m.

3.2 Data Standardization

The coordinates of the each of the cameras are stored as a $2 \times N$ vector for each cameras, stack together vertically, they formed the Row Data Matrix. However, the data is them process in a way the minimizes the errors of the SVD algorithm, and standardized so the mean of each row is located at 0 which is necessary for the theory to work.

Algorithm 3: Row Data Matrix Standardization

```
1: ScreenWidthHeightVec := [480; 640; 480; 640; 480; 640]
2: Input: RowDataMatrix: M
3: StdMatrix := M./mean(RowDataMatrix, 2)
4: StdMatrix := M./ScreenWidthHeightVec
5: Output: StdMatrix
```

Algorithm 3 standardizes row data matrix by subtracting the mean on each row by the mean of that row, and divides by the size of the screen. Vector representing the horizontal motion is divided by the width of the screen, and vector representing the vertical motion is divided by the height of the screen. This is necessary because the motion describes by vector is measured by screen size instead of pixels. The algorithm is implemented in HW3Demo.m

4 Computational Results

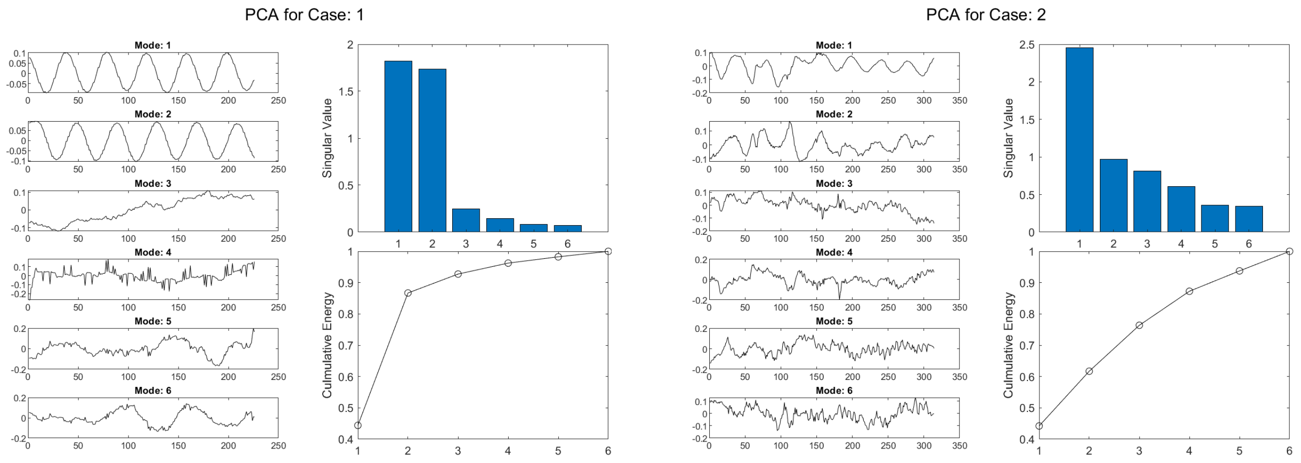

Figure 1: Harmonic Motion; Left: No Noise, Right: With Camera Vibrations as Noise

The ideal case is shown by the left of Figure 1. The modes are rows of the matrix V from SVD of the row data matrix. we will be able to obtain 2 harmonic oscillation that is slightly out of phase from each other. This indeed matches empirical observation from a human.

Computationally, the first 2 principal components contain over 90% of the cumulative energy. Compare to the case when noise are introduced as camera shakes, shown on right side of Figure 1, the cumulative

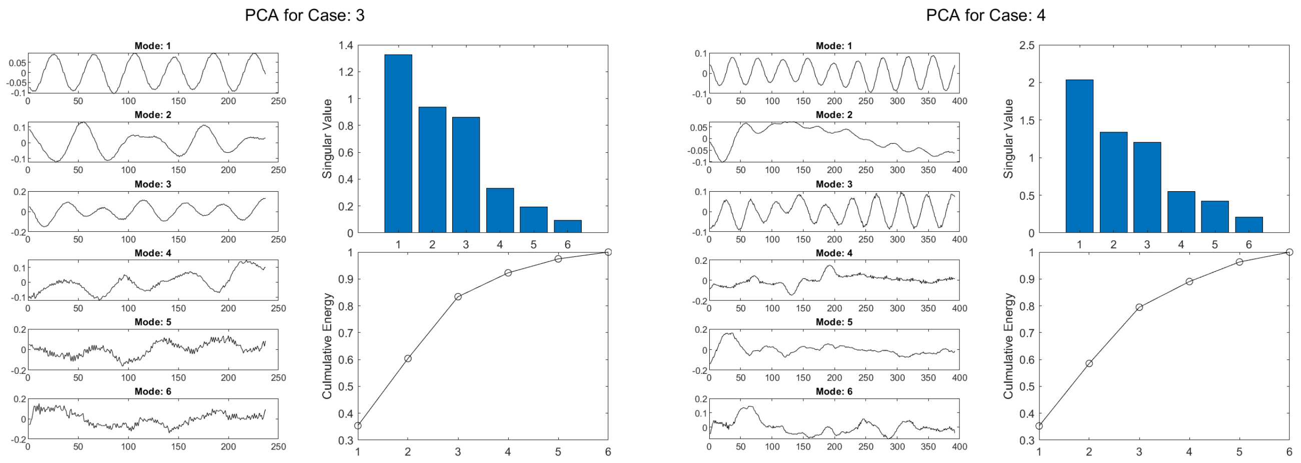

Figure 2: Left: Mixed Harmonic Pendulum Motion, Right: With Can Rotating

energy of the first 2 components decreased, but the first principal modes dramatically increased. In addition, the principal components describing the motions are also filled with noise.

The introduction of noise completely ruined the ability of the algorithm to identify low rank structure of observe motions.

Figure 2 left shows the PCA results for pendulum motions mixed with harmonic motions, which contains 3 principal components, each of them containing over 80% of the cumulative energy. The first component consists of motions from harmonic motions and the second and third consist of motions from the pendulum motion.

When noise are introduced for the mixed motions in the form of rotations, Algorithm 1 were not able to trace the motion for the can observed by the third camera due to rapid rotations of the can, causing distortion on the second principal component, this is shown the second mode plotted on the right of Figure 2.

5 Summary and Conclusions

Noise has a significant impact on the PCA algorithm, it peaks first singular value and reduces the precision of the principal components in when the noise is in the form of camera shakes. In the case rotation of the case, it's seemed as a deformation of the principal components. Once the type of noise is identified from the PCA components and distribution of the singular value, we then can apply better algorithms for motion extraction for noise reduction. It's not hard to notice that the changes between each frame should be used to track the motions of the can for each camera which will reduce the noise introduced by the can rotation. The noise introduce via camera shake can be reduces by actively tracking the motion of the whole frame. In addition to noise reduction techniques for the pre-processing of the signal, one can also use the ideas of Robust PCA [2].

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013, p. 391.

[2] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013, p. 403.

references

Appendix A MATLAB Functions

1. Tracer.m
   This is a class that contains all the parameters regarding the item, the frame, video, search range, etc. It stores them and performs the route tracing algorithm for the can in each video.

2. TracerSettings.m
   This is a function that stores all the settings for booting up the route trace algorithm. Things like the location of the can at the first frame, the size of the item frame, the search range etc.

3. Mono.m
   This function convert the RGB video to gray scale video.

4. FrameSearch.m
   This is a function that tries to minimize the difference between 2 matrices. Which is also the algorithm used to locate the can for each frame in the video.

5. HW3Demo.m
   This is the script that demonstrates all the procedures for looking for the can, PCA analysis, and plotting the results out.

   All the source codes used for this HW can be found in my git repository: Git repo

Tracer.m

```
classdef Tracer < handle
    % This is a class that stores the relavent parameters for a given video
    % It will call other static function to ahieve query about the given
    % video

    properties
        OriginalVid;    % The original video
        MonoVid;        % The video in mono.
        Item;           % The item in frame, which contain the object(the moving can) in mono vid.
        InitialPosition % The initial top top left corner of the frame.
        StartingPosition;
                        % Staring position, will get refreshed.
        SearchRange;    % The range to search for the next hot spot.
        Box;
    end

    methods
        function this = Tracer(video, box, startingPosi, searchRange)
            % video: w x d x 3 x f RGB video
            % box: w x d, the box containing the can on the mono vid
            % threshold: What kind of threshold for filtering out the
            % bright spot from the gray scale image.

            this.OriginalVid = video;
            this.MonoVid     = Mono(video);
            this.Item = this.MonoVid(startingPosi(1): startingPosi(1) + box(1),...
                                     startingPosi(2): startingPosi(2) + box(2), 1);
            this.Box = box;
            this.StartingPosition = startingPosi;
            this.InitialPosition = startingPosi;
            this.SearchRange = searchRange;
        end

        function fig = firstFrame(this)
           % Plot the first frame
           FirstFrameColored = this.OriginalVid(:, :, :, 1);
           FirstFrameMono = this.MonoVid(:, :, 1);
           fig = figure;
           x1 = this.InitialPosition(1);
           y1 = this.InitialPosition(2);
           x2 = x1 + this.Box(1) - 1; y2 = y1 + this.Box(2) - 1;
           subplot(2, 2, 1); imshow(FirstFrameColored);
           subplot(2, 2, 2); imshow(FirstFrameMono);
           subplot(2, 2, 3); imshow(FirstFrameColored(x1:x2, y1:y2, :, 1));
           subplot(2, 2, 4); imshow(FirstFrameMono(x1:x2, y1:y2, :, 1));
        end

        function coords = traceRoute(this)
            Video = this.MonoVid;
            coords = zeros(2, size(Video, 3));
            coords(:, 1) = [this.InitialPosition(1); this.InitialPosition(2)];
            for II = 2: size(Video, 3)
```

```
                Frame = double(Video(:, :, II));
                Displacement = FrameSearch(Frame, ...
                    this.Item, ...
                    this.StartingPosition(1), ...
                    this.StartingPosition(2), ...
                    this.SearchRange(1), ...
                    this.SearchRange(2)...
                    );

                disp(strcat("Frame: ", num2str(II), " Displacement: ", num2str(Displacement)));
                coords(:, II) = coords(:, II - 1) + Displacement';
                NewX = coords(1, II); NewY = coords(2, II);
                this.StartingPosition(1) = NewX;
                this.StartingPosition(2) = NewY;
                Height = size(this.Item, 1); Width = size(this.Item, 2);
                this.Item = Frame(NewX: NewX + Height - 1, NewY: NewY + Width - 1);
            end
        end


    end
end
```

TracerSettings.m

```
function searcher = TracerSettings(cam, instance)
   % Gave the camera, and the instance of the tesst, and the mono video
   % This functeion will return a tracer Setting
    loaded = load(strcat("data/cam", num2str(cam), "_", num2str(instance), ".mat"));
    originalVid = loaded.(strcat("vidFrames", num2str(cam), "_", num2str(instance)));
    switch cam
        case 1
            switch instance
                case 1
                    PositionX = 220;
                    PositionY = 310;
                    box = [80, 60];
                    threshold = 0.0;
                    searchRange = [30, 30];
                case 2
                    PositionX = 300;
                    PositionY = 310;
                    box = [80, 60];
                    threshold = 0.0;
                    searchRange = [30, 30];
                case 3
                    PositionX = 280;
                    PositionY = 310;
                    box = [50, 60];
                    threshold = 0.0;
                    searchRange = [30, 30];
                case 4
                    PositionX = 260;
                    PositionY = 370;
                    box = [60, 50];
                    threshold = 0.0;
                    searchRange = [30, 30];
            end
        case 2
            switch instance
                case 1
                    PositionX = 270;
                    PositionY = 260;
                    box = [90, 70];
                    threshold = 0.0;
                    searchRange = [30, 30];
                case 2
                    PositionX = 350;
                    PositionY = 290;
                    box = [50, 70];
                    threshold = 0.0;
                    searchRange = [50, 50];
                case 3
                    PositionX = 290;
                    PositionY = 220;
                    box = [80, 80];
                    threshold = 0.0;
                    searchRange = [50, 50];
                case 4
                    PositionX = 240;
                    PositionY = 220;
                    box = [100, 80];
                    threshold = 0.0;
                    searchRange = [50, 50];
            end
        case 3
            switch instance
                case 1
                    PositionX = 260;
                    PositionY = 310;
                    box = [60, 70];
                    threshold = 0.0;
                    searchRange = [30, 30];
                case 2
                    PositionX = 240;
                    PositionY = 340;
                    box = [50, 60];
                    threshold = 0.0;
                    searchRange = [30, 30];
                case 3
                    PositionX = 210;
                    PositionY = 340;
                    box = [60, 80];
                    threshold = 0.0;
                    searchRange = [30, 30];
                case 4
                    PositionX = 180;
                    PositionY = 350;
```

```
                    box = [60, 80];
                    threshold = 0.0;
                    searchRange = [30, 30];
            end
    end
    searcher = Tracer(originalVid, box, [PositionX, PositionY], searchRange);
end
```

Mono.m

```
function Converted = Mono(vid)
    % rgbVid:
    %   A 4D tensor representing a video, in the shape of w x d 3 x l,
    %   type: int8
    %
    % thresshold:
    %   A scaler in between 0 and 1 that filter out all piexels with an
    %   intensity that is less than it for the whole video

    Converted = zeros(size(vid, 1), size(vid, 2), size(vid, 4));
    for II = 1: size(vid, 4)
        Frame = vid(:, :, :, II);
        Frame = double(rgb2gray(Frame))/255; % to gray standardization
        Converted(:, :, II) = Frame;
    end
end
```

FrameSearch.m

```
function displacement = FrameSearch(background, searchFrame, positionX, ...
    positionY, rangeX, rangeY)
    %   This is a method that meant to be apply on images filter by the
    % ThresholdMono Function.
    %
    %   given a background, a searchFrame, a starting position positionX,
    % positionY, a range to search near it, rangeX, rangeY, the method will
    % slide the box around on the background and search for a displacement
    % vector such that the intensity is maximal.
    %
    % background:
    %   A matrix.
    % searchFrame: a vector denoting the width and height of the frame
    %   [width, height]
    % positionX, PositionY:
    %   The topleft corner of the search frame will be placed on this
    %   position.

    FrameH = size(searchFrame, 1);
    FrameW = size(searchFrame, 2);
    % Intensity = zeros(rangeX, rangeY);
    MinVal = inf;

    for II = -rangeX: rangeX - 1
        for JJ = -rangeY: rangeY - 1
```

```
            Xstart = positionX + II;
            Ystart = positionY + JJ;
            Xend = Xstart + FrameH - 1;
            Yend = Ystart + FrameW - 1;
            if Xstart < 1 || Ystart < 1 || Xend > size(background, 1) || Yend > size(background, 2)
                % ItemFrame Outside of the background, go back.
                continue;
            end
            SliceBackground = background(Xstart: Xend, Ystart: Yend);
            MinValCandidate = sum(abs(SliceBackground - searchFrame), "all");

            if MinValCandidate < MinVal
                displacement = [II, JJ];
                MinVal = MinValCandidate;
            end
        end
    end

end
```

HW3Demo.m

```
%% PCA ANALYSIS
clc; close all;

TEST_INSTANCE = 3;   % <== Tweak test instances here!

% Row data matrix format:
% cam 1, x, y coords on the 1, 2 row
% cam 2, x, y coords on the 3, 4 row
% cam 3, x, y coords on the 5, 6 row

RowDataMatrix = zeros(6, 1);
MinLength = inf;
for II = 0: 2
    tracer = TracerSettings(II + 1, TEST_INSTANCE);
    coords = tracer.traceRoute();
    RowDataMatrix(2*II + 1: 2*II + 2, 1: size(coords, 2)) = coords;
    MinLength = min(MinLength, size(coords, 2));
end
RowDataMatrix = RowDataMatrix(:, 1: MinLength);

%% DATA MATRIX VISUALIZE
figure(1);
sgtitle(strcat("Observed Data; Test Instance: ", num2str(TEST_INSTANCE)));
for II = 1: 3
    RowXIdx = (II - 1)*2 + 1;
    RowYIdx = RowXIdx + 1;

    subplot(3, 2, RowXIdx);
    plot(RowDataMatrix(RowXIdx, :), "k");
    ylabel("Pixels");
    xlabel("Frames");
    title(strcat("Cam", num2str(II), " vertical motion"));
    subplot(3, 2, RowYIdx);
    plot(RowDataMatrix(RowYIdx, :), "k");
    ylabel("Pixels");
    xlabel("Frames");
    title(strcat("Cam", num2str(II), " horizontal motion"));
end
saveas(gcf, strcat("observed-data-test-instance", num2str(TEST_INSTANCE)), "png");

%% PCA NORMALIZATION OVER THE WHOLE MATRIX

% Normalize motion wrt to the screen size capture by camera.
% Normalize the notion by subtracting the averae of each row.
ScreenSize = [480; 640; 480; 640; 480; 640];
StdDataMatrix = RowDataMatrix - mean(RowDataMatrix, 2);
StdDataMatrix = StdDataMatrix./ScreenSize;

figure(2);
title("Covariance on Std Data Matrix");
imagesc(cov(StdDataMatrix.'));
colorbar;
saveas(gcf, strcat("cov-std-matrix", num2str(TEST_INSTANCE)), "png");

[U, Sigma, V] = svd(StdDataMatrix, "econ");

figure('name', "PCA", 'Position', [0, 0, 900 600]);
sgtitle(strcat("PCA for Case: ", num2str(TEST_INSTANCE)))
for II = 1: 6
    subplot(6, 2, 2*(II - 1) + 1)
    plot(V(:, II), "k")
    title(strcat("Mode: ", num2str(II)))
end

subplot(6, 2, 2:2:6);
bar(1:6, diag(Sigma));
xlabel("Ranks of Singular Value");
ylabel("Singular Value");

subplot(6, 2, 8:2:12);
plot(cumsum(diag(Sigma))/sum(diag(Sigma)), "ko-");
ylabel("Cumulative Energy")
saveas(gcf, strcat("pca-analysis", num2str(TEST_INSTANCE)), "png");
```