Study of a Spring-Mass System with Principal Component Analysis

Arvindh Sharma *

February 22, 2020

Abstract

Principal Component Analysis (PCA) was used to identify the principal modes from a set of image data containing the dynamics of a spring-mass system. The mass, represented by a paint can, was located in the images using a novel algorithm. Singular Value Decomposition (SVD) was used to find the principal modes. It was found that PCA is effective in finding principal modes in datasets with large redundancy and less noise, while the effectiveness was reduced with the addition of noise. PCA was also found to require more than one mode to represent rotational motion.

1 Introduction and Overview

One of the major objectives of data analysis is to extract the dominant patterns from data. Real-world data is often of high-rank at first blush – however, they often represent systems that can be better represented by a fewer modes. Linear algebra, especially techniques of diagonalization, are often employed to achieve this. SVD and PCA are some of the important methods in diagonalizing higher-rank data to examine if the data could be better represented in fewer, more dominant, modes. We use the example of a simple spring-mass system, as observed by three cameras, to illustrate the principles involved in PCA.

2 Theoretical Background

PCA and SVD are transformations of data that allow us to represent higher dimensional data in terms of fewer, principal components, when there is redundancy in one or more modes. Here is a brief overview of the concepts [1].

2.1 Singular Value Decomposition (SVD)

SVD, similar to eigenvector decomposition, is a powerful method to transform a matrix into orthonormal bases. However, unlike eigenvector decomposition, it is always guaranteed to exist for all matrices, since it depends on two independent orthonormal bases instead of one. For a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, the SVD is given by Eq. (1), where $\boldsymbol{U}$ is a unitary transformation in $\mathbb{C}^{m\times m}$ (row-space), $\boldsymbol{V}$ is a unitary transformation in $\mathbb{C}^{n\times n}$ (column space), and $\boldsymbol{\Sigma}$ is a diagonal matrix in $\mathbb{R}^{m\times n}$.

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

Thus, the matrix $\boldsymbol{A}$ is decomposed into a rotation by the unitary matrix $\boldsymbol{V}^*$, then stretching by the diagonal matrix $\boldsymbol{\Sigma}$, and finally another rotation by the unitary matrix $\boldsymbol{U}$. While SVD has many mathematical properties that have wide applications in mathematics and science, one of the most important fact is given by the fact that SVD provides the best lower-rank approximation for a given higher-order system, as quantified by l^2 norm. This important result is encapsulated in Eq. (2), where $N \in [0, r]$, r is the rank of $\boldsymbol{A}$, σ_j are the diagonal values of $\boldsymbol{\Sigma}$, $\boldsymbol{u}_j$ are columns of $\boldsymbol{U}$, and $\boldsymbol{v}_j^*$ are columns of $\boldsymbol{V}^*$.

$$\boldsymbol{A}_N = \sum_{j=1}^{N} \sigma_j \boldsymbol{u}_j \boldsymbol{v}_j^* \quad . \tag{2}$$

*GitHub : `https://github.com/arvindhsharma`

2.2 Principal Component Analysis (PCA)

Principal Component Analysis is a variation on SVD, and allows us to study the important lower-order dynamics of a higher-dimensional data. This reduction in dimensionality is predicated on the assumption that the higher-rank data contains some redundancy, which can be removed to represent the data along fewer principal modes.

For instance, for a simple spring mass system undergoing simple harmonic motion in gravity, the governing equation is given by Eq. (3), where $f(t)$ is the position in vertical direction at time t, and ω is a constant.

$$\frac{d^2 f(t)}{dt^2} = -\omega^2 f(t) \quad . \tag{3}$$

The solution is given by Eq. (4), where a is the amplitude of the simple harmonic motion.

$$f(t) = acos(\omega t) \quad . \tag{4}$$

If this oscillatory motion is captured by a camera, however, the data will exist as co-ordinates in pixel space, which will be at least of two-dimensions in each image stack from a camera. Moreover, the camera might not be aligned perfectly with an axis of motion of the oscillating body, thus making it hard to deduce the motion from the pixel locations. There could be more than one camera recording the same object. If there is any motion at all in a direction other than the major oscillation, that will add to the complexity of the dynamics. If the camera is not held steady, this will add noise to the collected data as well.

This is where PCA can be effective: if the noise is not too high (based on signal-to-noise ratio), then the PCA can identify that the dynamics of the complex higher-order system might in fact be best represented by fewer principal modes. For instance, if $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_m$ are m vectors, of length n, representing the position of the oscillating body, then the data can be represented by the $m \times n$ matrix $\boldsymbol{X}$, as in Eq. (5).

$$\boldsymbol{X} = \begin{bmatrix} —\boldsymbol{x}_1— \\ —\boldsymbol{x}_2— \\ \vdots \\ —\boldsymbol{x}_m— \end{bmatrix} \quad . \tag{5}$$

From this, a symmetric $m \times m$ covariance matrix $\boldsymbol{C_X}$ can be formed, as shown in Eq. (6).

$$\boldsymbol{C_X} = \frac{1}{n-1} \boldsymbol{X} \boldsymbol{X}^T \quad . \tag{6}$$

In the covariance matrix, the diagonal terms are auto-correlations of the measured signals while the off-diagonal terms are correlations between different pairs of signals. For the diagonal correlation, a large value signifies that the associated measurement greatly influences the dynamics of the system. For off-diagonal correlation terms, a large value signifies that the pair of measurements in consideration are statistically similar, meaning there is redundancy. By corollary, a smaller correlation value here suggests statistical independence.

Diagonalizing the covariance matrix would yield variances along the major directions, or modes, along which the system's dynamics can be best described. Since $\boldsymbol{C_X}$ is real, square, and symmetric, eigenvector decompostion could be used. However, SVD is a more robust method to achieve this, and has the added benefit of sorting the diagonal matrix by largest variances. It can be shown that for the SVD transformation $\boldsymbol{X} = \boldsymbol{U\Sigma V^*}$, the square of the diagonal terms in $\boldsymbol{\Sigma}$ is the same as the eigenvalues of the covariance matrix $\boldsymbol{C_Y}$, where $\boldsymbol{Y} = \boldsymbol{U^* X}$ gives the projections of the data on to the principal modes. The columns of $\boldsymbol{V}_*$ give the dynamics of the system in time along the principal modes. Note that $\boldsymbol{X}$ in this case is a short-fat matrix, while we typically use tall-skinny matrices. In that case, the behavior of the unitary matrices $\boldsymbol{U}$ and $\boldsymbol{V}^*$ are reversed.

3 Algorithm Implementation and Development

Algorithm 1 provides a scheme to analyze the images to extract principal modes exhibited by the movement of the paint can. The process is repeated for each of the four cases under study. Here is a brief description of the process.

3.1 Data setup

Each case has images of the pain can's movement captured from three different camera locations, for a total of three image stacks per case. The image capture rate is the same for all the image stacks, and the distance of the camera to the paint can is roughly the same. The images are loaded as matrices of size $m \times n \times 3 \times T$, where $m \times n$ is the frame size of the RGB image and T is the number of frames. T varies from one image stack to another.

Algorithm 1 Algorithm to analyze the principal components in the motion of paint can.

Load the images from the three cameras
procedure DETECT POSITION OF PAINT CAN(image)
    $I\_r \leftarrow$ Set the pixels away from the paint can in the image to 0
    if Torch-light is visible in all frames **then**
        $I\_gs \leftarrow$ Convert RGB image to grayscale
        $I\_gs\_bl\_r \leftarrow$ Apply circular averaging filter to I_gs
        $I\_gs\_bl\_r\_d \leftarrow$ Pixels at a specific distance away from a guess of paint can location are set to 0
        $(x\_loc, y\_loc) \leftarrow$ Best estimate of location of brightest pixels in $I\_gs\_bl\_r\_d$ or I_gs
    else
        $pink\_filt \leftarrow$ All pixels except pink colored pixels in image are set to 0
        $pink\_filt\_d \leftarrow$ Pixels at a specific distance away from a guess of paint can location are set to 0
        $(x\_loc, y\_loc) \leftarrow$ Center point of non-zero pixels in $pink\_filt\_d$
    end if
end procedure
for $kk = 1:1:3$ **do** ▷ For each camera
    $(x\_loc(1), y\_loc(1)) \leftarrow$ Find the location of pain can in first image by inspection
    $frame\_num \leftarrow$ Number of images in the stack
    for $ii = 2:1:frame\_num$ **do**
        Run procedure in 2 to 14
        $(x\_loc(ii), y\_loc(ii)) \leftarrow$ result of procedure
    end for
end for
$X\_pca \leftarrow$ Matrix formed by vectors x_loc and y_loc from the 3 image stacks as rows
$X\_pca\_mod \leftarrow X\_pca-$ row-wise mean of X_pca
Perform SVD on covariance matrix of X_pca_mod and analyze the results

3.2 Tracking paint can in the three image stacks

A robust algorithm has been developed to track the position of the paint can from frame to frame. The first step is to use MATLAB's `implay()` function to play back the frames, and identify the pixel space where the paint can is restricted to in all the images. The area outside this space is set to zero intensity to reduce the region where the program has to look for the paint can. The location of the paint can in the first image is manually detected and recorded.

The next step is to find if the torch-light strapped to the top of the paint can is visible and bright in all the images. If it is, then the RGB image is first converted to grayscale. The grayscale image undergoes filtering with a circular averaging filter, since the torch-light is roughly circular in the images. Since the location of the paint can in the preceding image is always known, the pixels at a pre-determined distance away from that location is set to zero in the resulting filtered image. Then, the locations of the brightest pixel in the filtered image and the grayscale image are obtained. Using a custom condition for choosing which result to use based on the image stack (see Appendix B for individual conditions), the location of the paint can in the image is recorded.

If the torch-light is not visible in all the images, then we take advantage of the fact that the torch is made of a pink material, and no other object in the images close to the paint can is pink. From the RGB images, ratios of the red color to the green and blue colors are determined at each pixel in the area close to the paint can. Then the locations of the pink colored pixels in this restricted area are noted. From this pixel distribution, the first moment of the pixel locations in x and y directions provides a good estimate of the center of the torch. This point is then recorded as the location of the paint can, and is typically very close to the bright torch-light location. Both the methods described here are used in tandem whenever necessary. Figure 1 shows a montage of images, with the detected paint can location indicated by the red square.

3.3 Principal component analysis

The paint can locations are stored in vectors of length T. Since T varies from one image stack to the next, the vectors are arranged so that they are aligned with the paint can location in a similar point in its trajectory in each image stream. The length of vectors is also matched to a common size t. With the 6 vectors of length t, a matrix X_pca, of size $6 \times t$, is formed with the position vectors as its rows.

In order to use SVD to find the principal components, a covariance matrix is first formed from X_pca. This is done by first subtracting the mean of each row of X_pca, and subtracting this from each element of its corresponding row. Then, the matrix elements are divided by $t - 1$ to form the covariance matrix X_pca_mod. We then find the principal

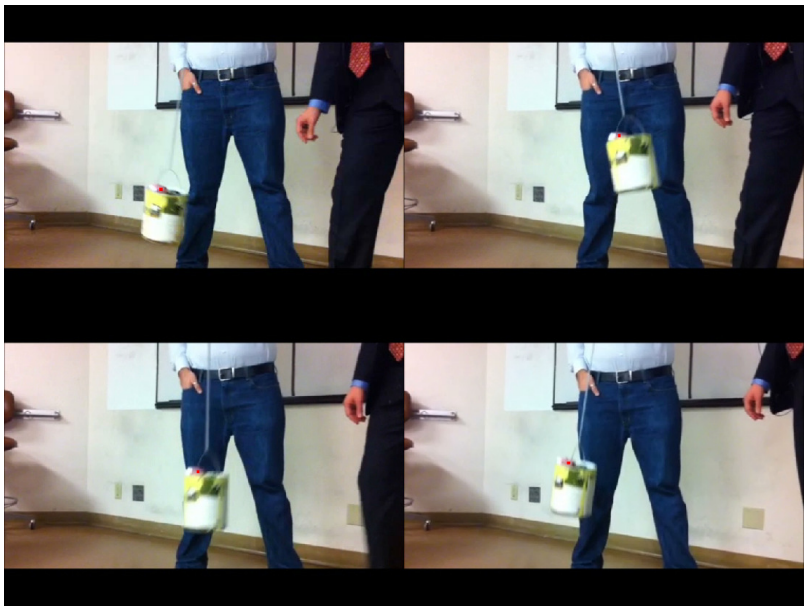

Figure 1: Montage showing images with the red square indicating the location of the torch-light, as determined by the algorithm.

modes and the projections of the pixel co-ordinates in principal modes by performing SVD on the resulting covariance matrix. Since the data here is on similar scale, standardization to get unit variance is not performed on the data.

4 Computational Results

4.1 Ideal case

In this case, the paint can is seen in the videos to be moving only up-and-down in the real world frame, with little to none camera shake. However, the camera frames are not aligned with the movement of the can and we need the PCA to help identify the major modes. After processing the images to obtain the paint can trajectory in the 3 image stacks, SVD is performed on the covariance matrix. Figure 2(d) shows the zero-mean, aligned position vectors gathered from the images. Note that the data requires 6 modes in this case to be effectively described, but we see that there might be some dominant modes. After performing SVD, the singular values, σ_k, and the cumulative energy in first singular values up to k, are plotted in Figure 2(a) and (b). It clearly confirms the intuition that only the first mode is dominant, as about 70% of the energy is contained in just the first mode. Mode 2 also has some associated energy, possibly due to the slight horizontal movement of the paint can.

Figure 2(c) is a plot of the first two dominant modes, showing how they evolve in time, while Figure 2(e) shows the projection of the data in the first two principal modes. The sinusoidal, oscillatory nature of the first mode and the projection of data on to it confirms our knowledge that the paint can's motion can be described well by an up-and-down movement alone along an axis.

4.2 Case with noisy data

In this case, we explore data similar to the previous ideal case, but with some camera shake. This is analagous to adding noise to the data from the ideal case. Figure 3(d) demonstrates this fact, and we see that all the signals from the positions obtained from the images are very noisy. Performing PCA, we find that the noise has artificially added more energy to modes other than the first mode, deviating from the ideal case, as can be seen in Figure 3(a) and (b). Similarly, Figure 3(c) and (e) shows that unlike in the ideal case, it is difficult to identify the smooth sinusoidal trend in the dominant modes. However, we can still see a noisy sinusoidal dominant mode, and more filtering techniques could potentially help extract a cleaner trend.

4.3 Case with horizontal displacement

The third case has the paint can undergoing simple harmonic motion against gravity in the vertical direction (z direction), while also swinging back and forth in the x-y plane. From the position data extracted from the images, as seen in Figure

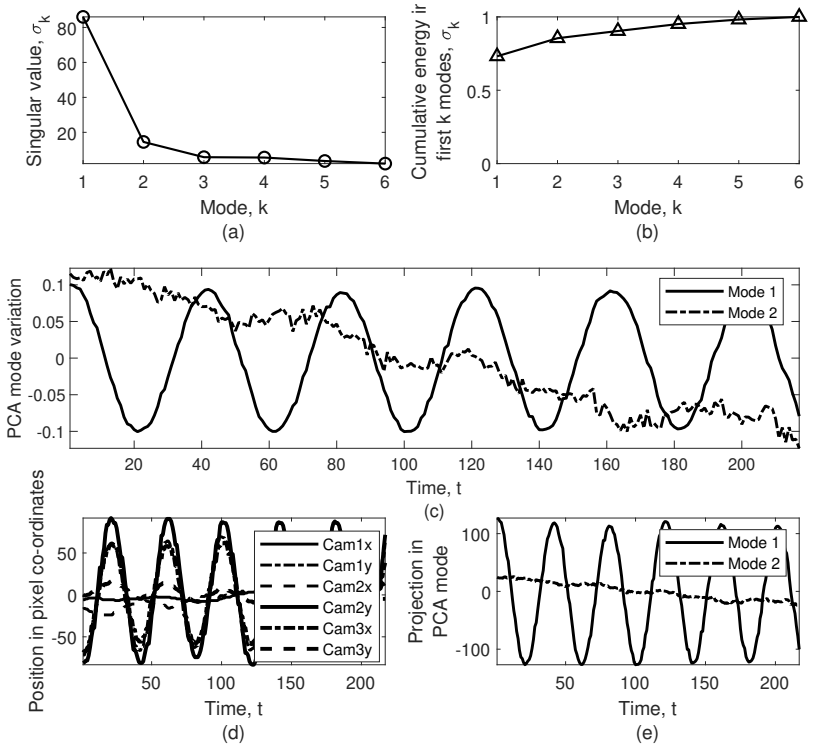

Figure 2: For the ideal case: (a) Plot showing the singular values of the modes; (b) Plot of energy contained in first k singular values; (c) Time-variation of the principal modes; (d) Time-series data of the paint can location detected from the images; (e) Projections of the data on the principal modes.

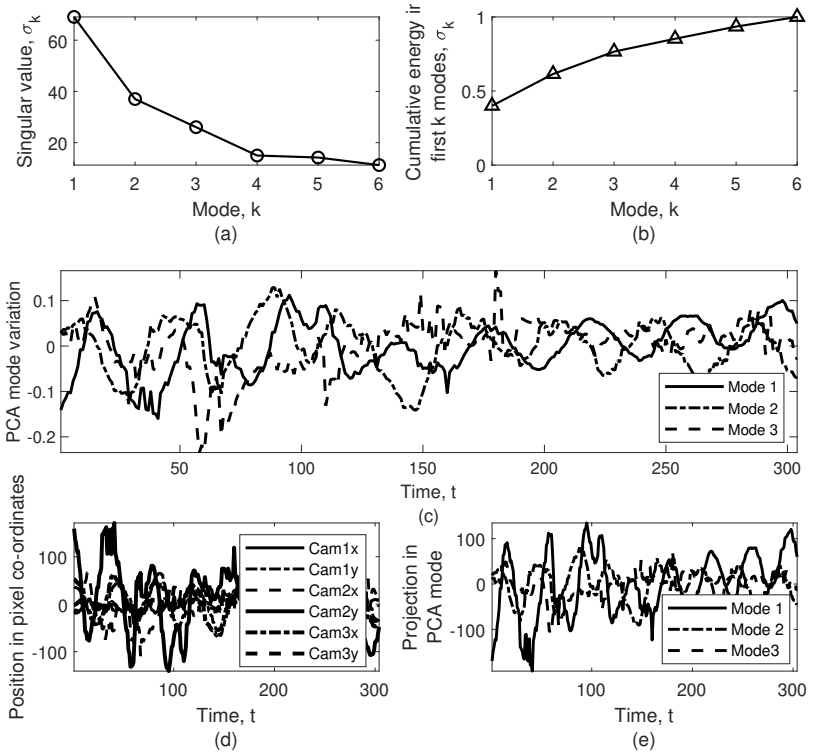

Figure 3: For the case with noisy data: (a) Plot showing the singular values of the modes; (b) Plot of energy contained in first k singular values; (c) Time-variation of the principal modes; (d) Time-series data of the paint can location detected from the images; (e) Projections of the data on the principal modes.

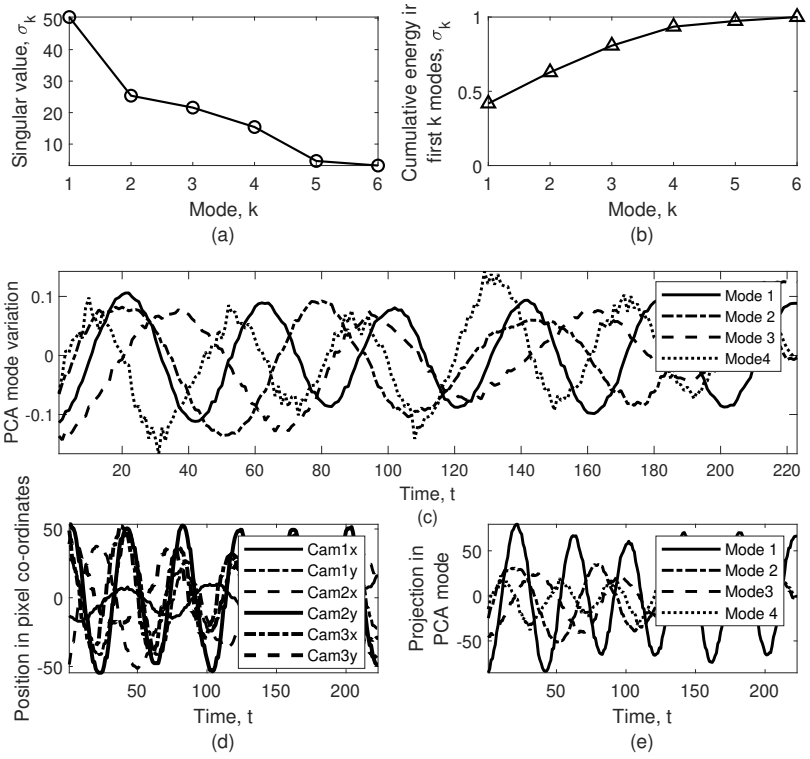

Figure 4: For the case with horizontal displacement: (a) Plot showing the singular values of the modes; (b) Plot of energy contained in first k singular values; (c) Time-variation of the principal modes; (d) Time-series data of the paint can location detected from the images; (e) Projections of the data on the principal modes.

4(d), it can be seen that the motion is registered in varying modes in the images. However, the results from SVD show that the motion can be better captured in first four modes (Figure 4(a) and (b)). The principal modes are all sinusoidal (Figure 4(c)), but we can from the projection of data on to the modes that mode 1 is much stronger than the rest (Figure 4(e)), and represents the vertical oscillations in the z direction from the images. Two of the other three modes represent the oscillation in x-y direction, and the other mode the movement perpendicular to the oscillations in mode 1.

4.4 Case with horizontal displacement and rotation

The final case has the paint can oscillating in the z direction, undergoing pendulum like motion in the x-y plane, while also rotating about its axis. Figure 5(d) shows these motions captured in the six modes from the camera. The results from PCA suggest that we need four principal modes to represent this motion (Figure 5(a) and (b)).

The principal mode is the up-and-down motion of the paint can, and the other three largest modes capture the oscillations in the x-y plane and the rotation motion. All of these motions are oscillations in their respective directions, and the sinusoidal modes and projections in Figure reffig:case4(c) and (e) illustrate this.

5 Summary and Conclusions

This work helps showcase the power of principal component analysis in picking the primary modes from higher dimensional data. From a series of images capturing the movement of a paint can from different angles, PCA is capable of picking the primary directions along which the motion of the paint can could be represented. By projecting the data onto the principal modes, we can find how the paint can moves along these principal directions. At the same time, the limitation of PCA are also evident from this work. For instance, adding noise to the data bleeds energy to more modes than necessary in the ideal case. Also, PCA is not very effective in representing rotational motion, and needs two principal directions instead of just one.

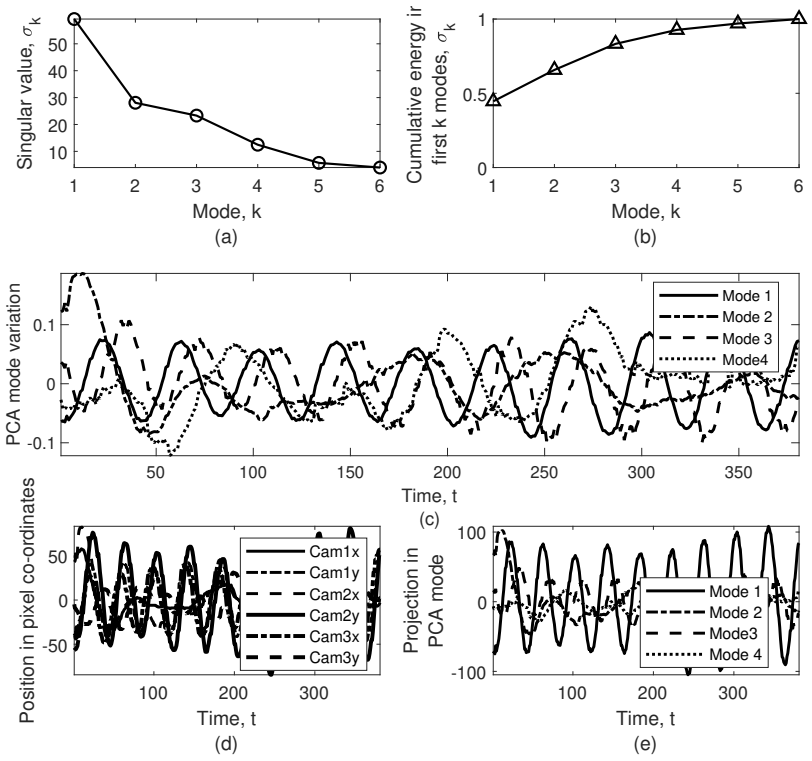

Figure 5: For the case with horizontal displacement and rotation: (a) Plot showing the singular values of the modes; (b) Plot of energy contained in first k singular values; (c) Time-variation of the principal modes; (d) Time-series data of the paint can location detected from the images; (e) Projections of the data on the principal modes.

References

[1] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019. DOI: 10.1017/9781108380690.

Appendix A MATLAB Functions

Here is an overview of the major functions used in the programs in Appendix B.

- `[U, S, V] = svd(A)` returns the singular value decomposition of the matrix `A`, such that `A = U * S * V'`.
- `implay(I, fps)` opens the Video Viewer app to view the image stack `I` as a video at the frame rate `fps`.
- `I = rgb2gray(RGB)` returns a grayscale equivalent of the color image `RGB`, where the equivalence is governed only by the luminance of pixels.
- `h = fspecial(type)` returns a 2-D filter of a select few types. For instance, `type` could be `average`, `disk`, `Gaussian`, and so on.
- `B = imfilter(A, h)` returns the result of filtering the multi-dimensional array `A` with the multi-dimensional filter `h`. Additional options can be provided to specify how to treat corner points.

Appendix B MATLAB Code

B.1 Ideal Case

```
%% LOADING DATA

clear; clc; close all
addpath('D:\course_work\amath582\hw3\data\')
```

```
C_1(3) = struct('data', [], 'results', []);
for jj = 1:1:3
    temp = load(strcat('cam', num2str(jj), '_1.mat'));
    temp1 = fieldnames(temp);
    C_1(jj).data = temp.(temp1{1});
end
clearvars temp

%% TRACKING THE PAINT CAN

for kk = 1:1:3

    sz = size(C_1(kk).data);
    C_1(kk).results.images = zeros(sz);
    C_1(kk).results.images = uint8(C_1(kk).results.images);
    x = 1:1:sz(2);
    y = 1:1:sz(1);
    num_of_frames = sz(end);
    [X, Y] = meshgrid(x, y);
    rec_mask = zeros(sz(1:2));
    if kk == 1
        rec_mask(X > 285 & X < 421) = 1;    % Restrict to interested area
    elseif kk == 2
        rec_mask(X > 220 & X < 383) = 1;
    else
        rec_mask(X > 221 & X < 469 & Y > 211 & Y < 335) = 1;
    end
    if kk ~= 3
        circ_filt = fspecial('disk', 4);    % A circular disk filter
    else
        circ_filt = fspecial('disk', 2);
    end
    C_1(kk).results.x_loc = nan(sz(end), 1);
    C_1(kk).results.y_loc = nan(sz(end), 1);
    figure, imshow(C_1(kk).data(:, :, :, 1))
    % Get initial (x, y) pixel location from first frame
    prompt1 = 'Enter x-coordinate of torchlight from the frame: ';
    C_1(kk).results.x_loc(1) = input(prompt1);
    prompt2 = 'Enter y-coordinate of torchlight from the frame: ';
    C_1(kk).results.y_loc(1) = input(prompt2);
    close;

    for ii = 2: 1: num_of_frames

        I = C_1(kk).data(:, :, :, ii);
        I_gs = rgb2gray(I);
        I_gs_bl = imfilter(double(I_gs), circ_filt, 0);
        I_gs_bl_r = I_gs_bl .* rec_mask;
        % dist_mask filters out pixels too far from previous known location
        dist_mask = sqrt((X - C_1(kk).results.x_loc(ii - 1)) .^ 2 +...
            (Y - C_1(kk).results.y_loc(ii - 1)) .^ 2);
        if kk ~= 3
            dist_mask(dist_mask < 19) = 1;
            dist_mask(dist_mask >= 19) = 0;
        else
            dist_mask(dist_mask < 30) = 1;
            dist_mask(dist_mask >= 30) = 0;
```

```
        end

        I_gs_bl_rd = I_gs_bl_r .* dist_mask;

        [~, maxpt_temp] = max(I_gs_bl_rd(:));

        I_gs_bl_rd_mod = I_gs_bl_rd;
        if kk ~= 3
            I_gs_bl_rd_mod(Y > (Y(maxpt_temp) - 8)) = 0;
        else
            I_gs_bl_rd_mod(X > (X(maxpt_temp) - 15)) = 0;
            I_gs_mod = double(I_gs) .* dist_mask .* rec_mask;
            I_gs_mod(X > (X(maxpt_temp) - 15)) = 0;
            [max_temp2, maxpt_temp2] = max(I_gs_mod(:));
        end

        [max_temp1, maxpt_temp1] = max(I_gs_bl_rd_mod(:));

        if max_temp1 > (.95 * 255)
            maxpt = maxpt_temp1;
        elseif kk == 3 && max_temp2 > (0.85 * 255)
            maxpt = maxpt_temp2;
        else
            maxpt = maxpt_temp;
        end

        % Shading the detected point in red
        temp_img = I;
        temp_img((Y(maxpt) - 2):(Y(maxpt) + 2),...
            (X(maxpt) - 2):(X(maxpt) + 2), 1) = 255;
        temp_img((Y(maxpt) - 2):(Y(maxpt) + 2),...
            (X(maxpt) - 2):(X(maxpt) + 2), 2:3) = 0;
        C_1(kk).results.images(:, :, :, ii) = temp_img;

        [C_1(kk).results.x_loc(ii, 1), C_1(kk).results.y_loc(ii, 1)] =...
            deal(X(maxpt), Y(maxpt));

    end

end

%% PCA AND PLOTTING

[min_len, min_len_case] = min([length(C_1(1).results.x_loc),...
    length(C_1(2).results.x_loc), length(C_1(3).results.x_loc)]);
new_len = length(C_1(1).results.x_loc(10:end));

X_pca = [C_1(1).results.x_loc(10:end)';
         C_1(1).results.y_loc(10:end)';
         C_1(2).results.x_loc(19:(19 + new_len - 1))';
         C_1(2).results.y_loc(19:(19 + new_len - 1))';
         C_1(3).results.x_loc(9:(9 + new_len - 1))';
         C_1(3).results.y_loc(9:(9 + new_len - 1))'];

X_pca_mod = X_pca - mean(X_pca, 2); % Standardizing
[U, S, V] = svd(X_pca_mod' / sqrt(new_len - 1));
sig = diag(S);
Y_pca = V' * X_pca_mod;
```

```
fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 6];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 6];

s1 = subplot(3, 2, 1);
s1.Box = 'on';
h1 = plot(sig, 'ko-', 'LineWidth', 1.1);
xlabel({'Mode, k'; '(a)'})
ylabel('Singular value, \sigma_k')
axis tight

s2 = subplot(3, 2, 2);
s2.Box = 'on';
h2 = plot(cumsum(sig) / sum(sig), 'k^-', 'LineWidth', 1.1);
xlabel({'Mode, k'; '(b)'})
ylabel({'Cumulative energy in', 'first k modes, \sigma_k'})
axis tight

s3 = subplot(3, 2, 3:4);
s3.Box = 'on';
hold on
h3 = plot(U(:, 1), 'k', 'LineWidth', 1.5);
h4 = plot(U(:, 2), 'k-.', 'LineWidth', 1.5);
legend('Mode 1', 'Mode 2')
axis tight
xlabel({'Time, t', '(c)'})
ylabel('PCA mode variation')

s4 = subplot(3, 2, 5);
s4.Box = 'on';
hold on
h5 = plot(X_pca_mod(1, :), 'k', 'LineWidth', 1.5);
h6 = plot(X_pca_mod(2, :), 'k-.', 'LineWidth', 1.5);
h7 = plot(X_pca_mod(3, :), 'k--', 'LineWidth', 1.5);
h8 = plot(X_pca_mod(4, :), 'k', 'LineWidth', 2);
h9 = plot(X_pca_mod(5, :), 'k-.', 'LineWidth', 2);
h10 = plot(X_pca_mod(6, :), 'k--', 'LineWidth', 2);
legend('Cam1x', 'Cam1y', 'Cam2x', 'Cam2y', 'Cam3x', 'Cam3y')
axis tight
xlabel({'Time, t', '(d)'})
ylabel('Position in pixel co-ordinates')

s5 = subplot(3, 2, 6);
s5.Box = 'on';
hold on
h11 = plot(Y_pca(1, :), 'k', 'LineWidth', 1.5);
h12 = plot(Y_pca(2, :), 'k-.', 'LineWidth', 1.5);
legend('Mode 1', 'Mode 2')
axis tight
xlabel({'Time, t', '(e)'})
ylabel({'Projection in', 'PCA mode'})
```

B.2 Noisy Case

```
%% LOADING DATA

clear; clc; close all
addpath('D:\course_work\amath582\hw3\data\')

C_2(3) = struct('data', [], 'results', []);
for jj = 1:1:3
    temp = load(strcat('cam', num2str(jj), '_2.mat'));
    temp1 = fieldnames(temp);
    C_2(jj).data = temp.(temp1{1});
end
clearvars temp

%% TRACKING THE PAINT CAN

for kk = 1:1:3

    sz = size(C_2(kk).data);
    C_2(kk).results.images = zeros(sz);
    C_2(kk).results.images = uint8(C_2(kk).results.images);
    x = 1:1:sz(2);
    y = 1:1:sz(1);
    num_of_frames = sz(end);
    [X, Y] = meshgrid(x, y);
    rec_mask = zeros(sz(1:2));
    if kk == 1
        rec_mask(X > 310 & X < 421) = 1;    % Restrict to interested area
    elseif kk == 2
        rec_mask(X > 180 & X < 400) = 1;
    else
        rec_mask(X > 221 & X < 469 & Y > 211 & Y < 335) = 1;
    end
    if kk ~= 3
        circ_filt = fspecial('disk', 3);    % A circular disk filter
    else
        circ_filt = fspecial('disk', 2);
    end
    C_2(kk).results.x_loc = nan(sz(end), 1);
    C_2(kk).results.y_loc = nan(sz(end), 1);
    figure, imshow(C_2(kk).data(:, :, :, 1))
    % Get initial (x, y) pixel location from first frame
    prompt1 = 'Enter x-coordinate of torchlight from the frame: ';
    C_2(kk).results.x_loc(1) = input(prompt1);
    prompt2 = 'Enter y-coordinate of torchlight from the frame: ';
    C_2(kk).results.y_loc(1) = input(prompt2);
    close;

    for ii = 2: 1: num_of_frames

        I = C_2(kk).data(:, :, :, ii);
        I_gs = rgb2gray(I);
        I_gs_bl = imfilter(double(I_gs), circ_filt, 0);
        I_gs_bl_r = I_gs_bl .* rec_mask;
        % dist_mask filters out pixels too far from previous known location
        dist_mask = sqrt((X - C_2(kk).results.x_loc(ii - 1)) .^ 2 +...
            (Y - C_2(kk).results.y_loc(ii - 1)) .^ 2);
        if kk == 1
```

```
            dist_mask(dist_mask < 55) = 1;
            dist_mask(dist_mask >= 55) = 0;
        elseif kk == 2
            dist_mask(dist_mask < 100) = 1;
            dist_mask(dist_mask >= 100) = 0;
        else
            dist_mask(dist_mask < 30) = 1;
            dist_mask(dist_mask >= 30) = 0;
        end

        I_gs_bl_rd = I_gs_bl_r .* dist_mask;

        [~, maxpt_temp] = max(I_gs_bl_rd(:));

        I_gs_bl_rd_mod = I_gs_bl_rd;
        if kk ~= 3
            I_gs_bl_rd_mod(Y > (Y(maxpt_temp) - 20)) = 0;
            I_gs_mod = double(I_gs) .* dist_mask .* rec_mask;
            I_gs_mod(Y > (Y(maxpt_temp) - 20)) = 0;
        else
            I_gs_bl_rd_mod(X > (X(maxpt_temp) - 15)) = 0;
            I_gs_mod = double(I_gs) .* dist_mask .* rec_mask;
            I_gs_mod(X > (X(maxpt_temp) - 15)) = 0;
        end

        [max_temp1, maxpt_temp1] = max(I_gs_bl_rd_mod(:));
        [max_temp2, maxpt_temp2] = max(I_gs_mod(:));


        if max_temp2 > (0.97 * 255)
            maxpt = maxpt_temp2;
        elseif max_temp1 > (.95 * 255)
            maxpt = maxpt_temp1;
        else
            maxpt = maxpt_temp;
        end

        % Shading the detected point in red
        temp_img = I;
        temp_img((Y(maxpt) - 2):(Y(maxpt) + 2),...
            (X(maxpt) - 2):(X(maxpt) + 2), 1) = 255;
        temp_img((Y(maxpt) - 2):(Y(maxpt) + 2),...
            (X(maxpt) - 2):(X(maxpt) + 2), 2:3) = 0;
        C_2(kk).results.images(:, :, :, ii) = temp_img;

        [C_2(kk).results.x_loc(ii, 1), C_2(kk).results.y_loc(ii, 1)] =...
            deal(X(maxpt), Y(maxpt));

    end

end

%% PCA AND PLOTTING

[min_len, min_len_case] = min([length(C_2(1).results.x_loc),...
    length(C_2(2).results.x_loc), length(C_2(3).results.x_loc)]);
new_len = length(C_2(1).results.x_loc(11:end));
```

```
X_pca = [C_2(1).results.x_loc(11:end)';
         C_2(1).results.y_loc(11:end)';
         C_2(2).results.x_loc(3:(3 + new_len - 1))';
         C_2(2).results.y_loc(3:(3 + new_len - 1))';
         C_2(3).results.x_loc(17:(17 + new_len - 1))';
         C_2(3).results.y_loc(17:(17 + new_len - 1))'];

X_pca_mod = X_pca - mean(X_pca, 2); % Standardizing
[U, S, V] = svd(X_pca_mod' / sqrt(new_len - 1));
sig = diag(S);
Y_pca = V' * X_pca_mod;

fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 6];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 6];

s1 = subplot(3, 2, 1);
s1.Box = 'on';
h1 = plot(sig, 'ko-', 'LineWidth', 1.1);
xlabel({'Mode, k'; '(a)'})
ylabel('Singular value, \sigma_k')
axis tight

s2 = subplot(3, 2, 2);
s2.Box = 'on';
h2 = plot(cumsum(sig) / sum(sig), 'k^-', 'LineWidth', 1.1);
xlabel({'Mode, k'; '(b)'})
ylabel({'Cumulative energy in', 'first k modes, \sigma_k'})
axis tight

s3 = subplot(3, 2, 3:4);
s3.Box = 'on';
hold on
h3 = plot(U(:, 1), 'k', 'LineWidth', 1.5);
h4 = plot(U(:, 2), 'k-.', 'LineWidth', 1.5);
h4a = plot(U(:, 3), 'k--', 'LineWidth', 1.5);
legend('Mode 1', 'Mode 2', 'Mode 3')
axis tight
xlabel({'Time, t', '(c)'})
ylabel('PCA mode variation')

s4 = subplot(3, 2, 5);
s4.Box = 'on';
hold on
h5 = plot(X_pca_mod(1, :), 'k', 'LineWidth', 1.5);
h6 = plot(X_pca_mod(2, :), 'k-.', 'LineWidth', 1.5);
h7 = plot(X_pca_mod(3, :), 'k--', 'LineWidth', 1.5);
h8 = plot(X_pca_mod(4, :), 'k', 'LineWidth', 2);
h9 = plot(X_pca_mod(5, :), 'k-.', 'LineWidth', 2);
h10 = plot(X_pca_mod(6, :), 'k--', 'LineWidth', 2);
legend('Cam1x', 'Cam1y', 'Cam2x', 'Cam2y', 'Cam3x', 'Cam3y')
axis tight
xlabel({'Time, t', '(d)'})
ylabel('Position in pixel co-ordinates')

s5 = subplot(3, 2, 6);
```

```
s5.Box = 'on';
hold on
h11 = plot(Y_pca(1, :), 'k', 'LineWidth', 1.5);
h12 = plot(Y_pca(2, :), 'k-.', 'LineWidth', 1.5);
h12a = plot(Y_pca(3, :), 'k--', 'LineWidth', 1.5);
legend('Mode 1', 'Mode 2', 'Mode3')
axis tight
xlabel({'Time, t', '(e)'})
ylabel({'Projection in', 'PCA mode'})
```

B.3 Horizontal Displacement Case

```
%% LOADING DATA

clear; clc; close all
addpath('D:\course_work\amath582\hw3\data\')

C_3(3) = struct('data', [], 'results', []);
for jj = 1:1:3
    temp = load(strcat('cam', num2str(jj), '_3.mat'));
    temp1 = fieldnames(temp);
    C_3(jj).data = temp.(temp1{1});
end
clearvars temp

%% TRACKING THE PAINT CAN

for kk = 1:1:3

    sz = size(C_3(kk).data);
    C_3(kk).results.images = zeros(sz);
    C_3(kk).results.images = uint8(C_3(kk).results.images);
    x = 1:1:sz(2);
    y = 1:1:sz(1);
    num_of_frames = sz(end);
    [X, Y] = meshgrid(x, y);
    rec_mask = zeros(sz(1:2));
    if kk == 1
        rec_mask(X > 270 & X < 380) = 1;    % Restrict to interested area
    elseif kk == 2
        rec_mask(X > 220 & X < 390) = 1;
    else
        rec_mask(X > 260 & X < 425 & Y > 185 & Y < 320) = 1;
    end
%     if kk ~= 3
%         circ_filt = fspecial('disk', 3);
    if kk == 3
        circ_filt = fspecial('disk', 2);    % A circular disk filter
    end
    C_3(kk).results.x_loc = nan(sz(end), 1);
    C_3(kk).results.y_loc = nan(sz(end), 1);
    figure, imshow(C_3(kk).data(:, :, :, 1))
    % Get initial (x, y) pixel location from first frame
    prompt1 = 'Enter x-coordinate of torchlight from the frame: ';
    C_3(kk).results.x_loc(1) = input(prompt1);
    prompt2 = 'Enter y-coordinate of torchlight from the frame: ';
    C_3(kk).results.y_loc(1) = input(prompt2);
    close;
```

```
for ii = 2: 1: num_of_frames

    I = C_3(kk).data(:, :, :, ii);
    if kk ~= 3
        I_r = double(I) .* repmat(rec_mask, [1, 1, 3]);
        r2g = I_r(:, :, 1) ./ I_r(:, :, 2);
        r2b = I_r(:, :, 1) ./ I_r(:, :, 3);
        % pink_filt looks for pink colored pixels
        pink_filt = (r2g > 1.2 & r2g < 1.55 & r2b > 1.1 & r2b < 1.5);
    else
        I_gs = rgb2gray(I);
        I_gs_bl = imfilter(double(I_gs), circ_filt, 0);
        I_gs_bl_r = I_gs_bl .* rec_mask;
    end
    % dist_mask filters out pixels too far from previous known location
    dist_mask = sqrt((X - C_3(kk).results.x_loc(ii - 1)) .^ 2 +...
        (Y - C_3(kk).results.y_loc(ii - 1)) .^ 2);
    if kk == 1
        dist_mask(dist_mask < 20) = 1;
        dist_mask(dist_mask >= 20) = 0;
    elseif kk == 2
        dist_mask(dist_mask < 30) = 1;
        dist_mask(dist_mask >= 30) = 0;
    else
        dist_mask(dist_mask < 20) = 1;
        dist_mask(dist_mask >= 20) = 0;
    end

    if kk ~= 3
        pink_filt_d = pink_filt .* dist_mask;
        x_pt = round(sum(sum(pink_filt_d .* X)) / sum(pink_filt_d(:)));
        y_pt = round(sum(sum(pink_filt_d .* Y)) / sum(pink_filt_d(:)));
    else
        I_gs_bl_rd = I_gs_bl_r .* dist_mask;
        [~, maxpt_temp] = max(I_gs_bl_rd(:));

        I_gs_bl_rd_mod = I_gs_bl_rd;
        I_gs_bl_rd_mod(X > (X(maxpt_temp) - 15)) = 0;
        I_gs_mod = double(I_gs) .* dist_mask .* rec_mask;
        I_gs_mod(X > (X(maxpt_temp) - 15)) = 0;

        [max_temp1, maxpt_temp1] = max(I_gs_bl_rd_mod(:));
        [max_temp2, maxpt_temp2] = max(I_gs_mod(:));


        if max_temp2 > (0.97 * 255)
            maxpt = maxpt_temp2;
        elseif max_temp1 > (.95 * 255)
            maxpt = maxpt_temp1;
        else
            maxpt = maxpt_temp;
        end
        [x_pt, y_pt] = deal(X(maxpt), Y(maxpt));
    end

    % Shading the detected point in red
    temp_img = I;
```

```
        temp_img((y_pt - 2):(y_pt + 2),...
            (x_pt - 2):(x_pt + 2), 1) = 255;
        temp_img((y_pt - 2):(y_pt + 2),...
            (x_pt - 2):(x_pt + 2), 2:3) = 0;
        C_3(kk).results.images(:, :, :, ii) = temp_img;

        [C_3(kk).results.x_loc(ii, 1), C_3(kk).results.y_loc(ii, 1)] =...
            deal(x_pt, y_pt);

    end

end

%% MONTAGE

multi = cat(4, C_3(2).results.images(:, :, :, 2:18:56));
fig1 = figure;
montage(multi)
fig1.Units = 'inches';
fig1.Position = [-.1 1.8 6.75 5.0625];
fig1.PaperUnits = 'inches';
fig1.PaperSize = [6.75 5.0625];

%% PCA AND PLOTTING

[min_len, min_len_case] = min([length(C_3(1).results.x_loc),...
    length(C_3(2).results.x_loc), length(C_3(3).results.x_loc)]);
% new_len = length(C_3(3).results.x_loc(13:end));
new_len = 223;

X_pca = [C_3(1).results.x_loc(17:(17 + new_len - 1))';
         C_3(1).results.y_loc(17:(17 + new_len - 1))';
         C_3(2).results.x_loc(4:(4 + new_len - 1))';
         C_3(2).results.y_loc(4:(4 + new_len - 1))';
         C_3(3).results.x_loc(13:(13 + new_len - 1))';
         C_3(3).results.y_loc(13:(13 + new_len - 1))'];

X_pca_mod = X_pca - mean(X_pca, 2); % Standardizing
[U, S, V] = svd(X_pca_mod' / sqrt(new_len - 1));
sig = diag(S);
Y_pca = V' * X_pca_mod;

fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 6];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 6];

s1 = subplot(3, 2, 1);
s1.Box = 'on';
h1 = plot(sig, 'ko-', 'LineWidth', 1.1);
xlabel({'Mode, k'; '(a)'})
ylabel('Singular value, \sigma_k')
axis tight

s2 = subplot(3, 2, 2);
s2.Box = 'on';
h2 = plot(cumsum(sig) / sum(sig), 'k^-', 'LineWidth', 1.1);
```

```
xlabel({'Mode, k'; '(b)'})
ylabel({'Cumulative energy in', 'first k modes, \sigma_k'})
axis tight

s3 = subplot(3, 2, 3:4);
s3.Box = 'on';
hold on
h3 = plot(U(:, 1), 'k', 'LineWidth', 1.5);
h4 = plot(U(:, 2), 'k-.', 'LineWidth', 1.5);
h4a = plot(U(:, 3), 'k--', 'LineWidth', 1.5);
h4b = plot(U(:, 4), 'k:', 'LineWidth', 1.5);
legend('Mode 1', 'Mode 2', 'Mode 3', 'Mode4')
axis tight
xlabel({'Time, t', '(c)'})
ylabel('PCA mode variation')

s4 = subplot(3, 2, 5);
s4.Box = 'on';
hold on
h5 = plot(X_pca_mod(1, :), 'k', 'LineWidth', 1.5);
h6 = plot(X_pca_mod(2, :), 'k-.', 'LineWidth', 1.5);
h7 = plot(X_pca_mod(3, :), 'k--', 'LineWidth', 1.5);
h8 = plot(X_pca_mod(4, :), 'k', 'LineWidth', 2);
h9 = plot(X_pca_mod(5, :), 'k-.', 'LineWidth', 2);
h10 = plot(X_pca_mod(6, :), 'k--', 'LineWidth', 2);
legend('Cam1x', 'Cam1y', 'Cam2x', 'Cam2y', 'Cam3x', 'Cam3y')
axis tight
xlabel({'Time, t', '(d)'})
ylabel('Position in pixel co-ordinates')

s5 = subplot(3, 2, 6);
s5.Box = 'on';
hold on
h11 = plot(Y_pca(1, :), 'k', 'LineWidth', 1.5);
h12 = plot(Y_pca(2, :), 'k-.', 'LineWidth', 1.5);
h12a = plot(Y_pca(3, :), 'k--', 'LineWidth', 1.5);
h12b = plot(Y_pca(4, :), 'k:', 'LineWidth', 1.5);
legend('Mode 1', 'Mode 2', 'Mode3', 'Mode 4')
axis tight
xlabel({'Time, t', '(e)'})
ylabel({'Projection in', 'PCA mode'})
```

B.4 Horizontal Displacement with Rotation Case

```
%% LOADING DATA

clear; clc; close all
addpath('D:\course_work\amath582\hw3\data\')

C_4(3) = struct('data', [], 'results', []);
for jj = 1:1:3
    temp = load(strcat('cam', num2str(jj), '_4.mat'));
    temp1 = fieldnames(temp);
    C_4(jj).data = temp.(temp1{1});
end
clearvars temp

%% TRACKING THE PAINT CAN
```

```
for kk = 1:1:3

    sz = size(C_4(kk).data);
    C_4(kk).results.images = zeros(sz);
    C_4(kk).results.images = uint8(C_4(kk).results.images);
    x = 1:1:sz(2);
    y = 1:1:sz(1);
    num_of_frames = sz(end);
    [X, Y] = meshgrid(x, y);
    rec_mask = zeros(sz(1:2));
    if kk == 1
        rec_mask(X > 310 & X < 470 & Y > 205 & Y < 361) = 1;
    elseif kk == 2
        rec_mask(X > 220 & X < 408) = 1;    % Restrict to interested area
    else
        rec_mask(X > 260 & X < 500 & Y > 134 & Y < 284) = 1;
    end
    if kk ~= 1
        circ_filt = fspecial('disk', 2);    % A circular disk filter
    end
    C_4(kk).results.x_loc = nan(sz(end), 1);
    C_4(kk).results.y_loc = nan(sz(end), 1);
    figure, imshow(C_4(kk).data(:, :, :, 1))
    % Get initial (x, y) pixel location from first frame
    prompt1 = 'Enter x-coordinate of torchlight from the frame: ';
    C_4(kk).results.x_loc(1) = input(prompt1);
    prompt2 = 'Enter y-coordinate of torchlight from the frame: ';
    C_4(kk).results.y_loc(1) = input(prompt2);
    close;

    for ii = 2: 1: num_of_frames

        I = C_4(kk).data(:, :, :, ii);
        I_r = double(I) .* repmat(rec_mask, [1, 1, 3]);
        r2g = I_r(:, :, 1) ./ I_r(:, :, 2);
        r2b = I_r(:, :, 1) ./ I_r(:, :, 3);
        g2b = I_r(:, :, 2) ./ I_r(:, :, 3);
        if kk == 1
            % pink_filt looks for pink colored pixels
            pink_filt = (r2g > 1.1 & r2g < 1.55 & r2b > 1.1 & r2b < 1.5...
                & I_r(:, :, 1) > 175 & g2b > .89 & g2b < 1.05);
        else
            pink_filt = (r2g > 1.1 & r2g < 1.55 & r2b > 1.1 & r2b < 1.5);
            I_gs = rgb2gray(I);
            I_gs_bl = imfilter(double(I_gs), circ_filt, 0);
            I_gs_bl_r = I_gs_bl .* rec_mask;
        end
        % dist_mask filters out pixels too far from previous known location
        dist_mask = sqrt((X - C_4(kk).results.x_loc(ii - 1)) .^ 2 +...
            (Y - C_4(kk).results.y_loc(ii - 1)) .^ 2);
        if kk == 1
            dist_mask(dist_mask < 13) = 1;
            dist_mask(dist_mask >= 13) = 0;
        elseif kk == 2
            dist_mask(dist_mask < 20) = 1;
            dist_mask(dist_mask >= 20) = 0;
        else
```

```
            dist_mask(dist_mask < 20) = 1;
            dist_mask(dist_mask >= 20) = 0;
        end

        pink_filt_d = pink_filt .* dist_mask;
        x_pt = round(sum(sum(pink_filt_d .* X)) / sum(pink_filt_d(:)));
        y_pt = round(sum(sum(pink_filt_d .* Y)) / sum(pink_filt_d(:)));

        if isnan(x_pt) || isnan(y_pt)
            I_gs_bl_rd = I_gs_bl_r .* dist_mask;
            [~, maxpt_temp] = max(I_gs_bl_rd(:));
            maxpt = maxpt_temp;
            [x_pt, y_pt] = deal(X(maxpt), Y(maxpt));
        end

        % Shading the detected point in red
        temp_img = I;
        temp_img((y_pt - 2):(y_pt + 2),...
            (x_pt - 2):(x_pt + 2), 1) = 255;
        temp_img((y_pt - 2):(y_pt + 2),...
            (x_pt - 2):(x_pt + 2), 2:3) = 0;
        C_4(kk).results.images(:, :, :, ii) = temp_img;

        [C_4(kk).results.x_loc(ii, 1), C_4(kk).results.y_loc(ii, 1)] =...
            deal(x_pt, y_pt);
    end

end

%% PCA AND PLOTTING

[min_len, min_len_case] = min([length(C_4(1).results.x_loc),...
    length(C_4(2).results.x_loc), length(C_4(3).results.x_loc)]);
new_len = length(C_4(1).results.x_loc(12:end));

X_pca = [C_4(1).results.x_loc(12:end)';
         C_4(1).results.y_loc(12:end)';
         C_4(2).results.x_loc(16:(16 + new_len - 1))';
         C_4(2).results.y_loc(16:(16 + new_len - 1))';
         C_4(3).results.x_loc(14:(14 + new_len - 1))';
         C_4(3).results.y_loc(14:(14 + new_len - 1))'];


X_pca_mod = X_pca - mean(X_pca, 2); % Standardizing
[U, S, V] = svd(X_pca_mod' / sqrt(new_len - 1));
sig = diag(S);
Y_pca = V' * X_pca_mod;

fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 6];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 6];

s1 = subplot(3, 2, 1);
s1.Box = 'on';
h1 = plot(sig, 'ko-', 'LineWidth', 1.1);
xlabel({'Mode, k'; '(a)'})
```

```
ylabel('Singular value, \sigma_k')
axis tight

s2 = subplot(3, 2, 2);
s2.Box = 'on';
h2 = plot(cumsum(sig) / sum(sig), 'k^-', 'LineWidth', 1.1);
xlabel({'Mode, k'; '(b)'})
ylabel({'Cumulative energy in', 'first k modes, \sigma_k'})
axis tight

s3 = subplot(3, 2, 3:4);
s3.Box = 'on';
hold on
h3 = plot(U(:, 1), 'k', 'LineWidth', 1.5);
h4 = plot(U(:, 2), 'k-.', 'LineWidth', 1.5);
h4a = plot(U(:, 3), 'k--', 'LineWidth', 1.5);
h4b = plot(U(:, 4), 'k:', 'LineWidth', 1.5);
legend('Mode 1', 'Mode 2', 'Mode 3', 'Mode4')
axis tight
xlabel({'Time, t', '(c)'})
ylabel('PCA mode variation')

s4 = subplot(3, 2, 5);
s4.Box = 'on';
hold on
h5 = plot(X_pca_mod(1, :), 'k', 'LineWidth', 1.5);
h6 = plot(X_pca_mod(2, :), 'k-.', 'LineWidth', 1.5);
h7 = plot(X_pca_mod(3, :), 'k--', 'LineWidth', 1.5);
h8 = plot(X_pca_mod(4, :), 'k', 'LineWidth', 2);
h9 = plot(X_pca_mod(5, :), 'k-.', 'LineWidth', 2);
h10 = plot(X_pca_mod(6, :), 'k--', 'LineWidth', 2);
legend('Cam1x', 'Cam1y', 'Cam2x', 'Cam2y', 'Cam3x', 'Cam3y')
axis tight
xlabel({'Time, t', '(d)'})
ylabel('Position in pixel co-ordinates')

s5 = subplot(3, 2, 6);
s5.Box = 'on';
hold on
h11 = plot(Y_pca(1, :), 'k', 'LineWidth', 1.5);
h12 = plot(Y_pca(2, :), 'k-.', 'LineWidth', 1.5);
h12a = plot(Y_pca(3, :), 'k--', 'LineWidth', 1.5);
h12b = plot(Y_pca(4, :), 'k:', 'LineWidth', 1.5);
legend('Mode 1', 'Mode 2', 'Mode3', 'Mode 4')
axis tight
xlabel({'Time, t', '(e)'})
ylabel({'Projection in', 'PCA mode'})
```