Principal Component Analysis of Videos of a Moving Mass

Ashley Batchelor

Abstract

Principal Component Analysis (PCA) was used to analyze three sets of video frames of a moving mass under four different conditions: an ideal case, a noisy case, a case with horizontal motion, and a case with horizontal motion and rotation.

Sec. I. Introduction and Overview

Principal Component Analysis is one of the most useful applications of Singular Value Decomposition (SVD). For this experiment, I analyzed three videos of four cases of motion of a paint can mass on a spring. The first case was an ideal case with only vertical motion. The second case included a significant amount of noise in the videos due to camera shake. The third case included horizontal motion, i.e. one more degree of freedom in motion compared to the ideal case. The fourth case included horizontal motion and rotational motion, i.e. two more degrees of freedom in motion compared to the ideal case.

Sec. II. Theoretical Background

A mass hanging from a spring, displaced in a vertical direction from its equilibrium position may be modeled by the simple differential equation.

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

This has the simple harmonic oscillator solution:

$$f(t) = A\, cos(\omega t + \phi_0)$$

Where the amplitude A, the angular frequency ω, and the initial phase ϕ_0 are determined by the initial displacement of the mass and the stiffness of the spring.

This motion may be measured by a camera as a function of two vectors x and y on the pixel array of the camera. Additional degrees of freedom may affect the motion of the mass. For example, pendulum motion (horizontal motion) and a rotational motion. Multiple cameras may be used to image the motion and from the data collected by these cameras, a common representation of the motion may be generated using SVD. The data from the three cameras may be represented in a single matrix:

$$X = \begin{bmatrix} \boldsymbol{x_a} \\ \boldsymbol{y_a} \\ \boldsymbol{x_b} \\ \boldsymbol{y_b} \\ \boldsymbol{x_c} \\ \boldsymbol{y_c} \end{bmatrix}$$

Performing SVD on this matrix X provides the matrices U, Σ, and V.[1] From these matrices, one can determine a set of principal components and eliminate or reduce redundant information about the motion of the mass.

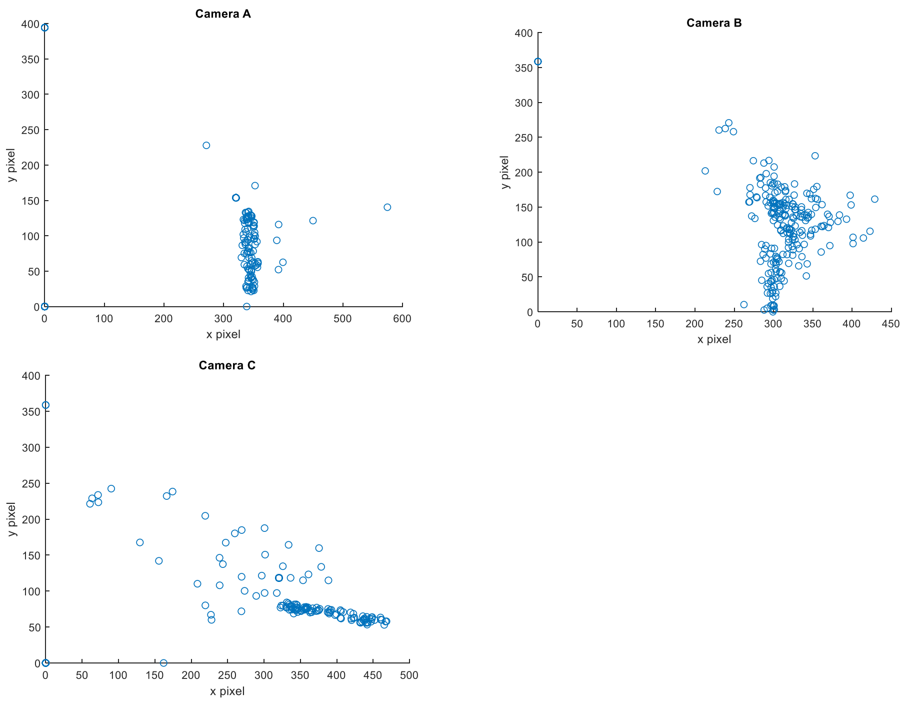
Sec. III. Algorithm Implementation and Development

For each of the four cases, I started by applying a motion tracking algorithm to each set of video frames in order to find a centroid of the paint can object. I synchronized the relative motion of the video frames by finding a minimum of the y displacement. I then determined respective vectors x_a, y_a, x_b, y_b, x_c, and $y_c$representing the pixel positions (x,y) of the centroid of the paint can as measured by each of the three cameras denoted a, b, and c. From these respective vectors, I generated the final matrix X and applied SVD to the matrix X, which returned the matrices U, Σ, and V. From the matrix Σ, I determined the principal components of motion for each of the four cases.

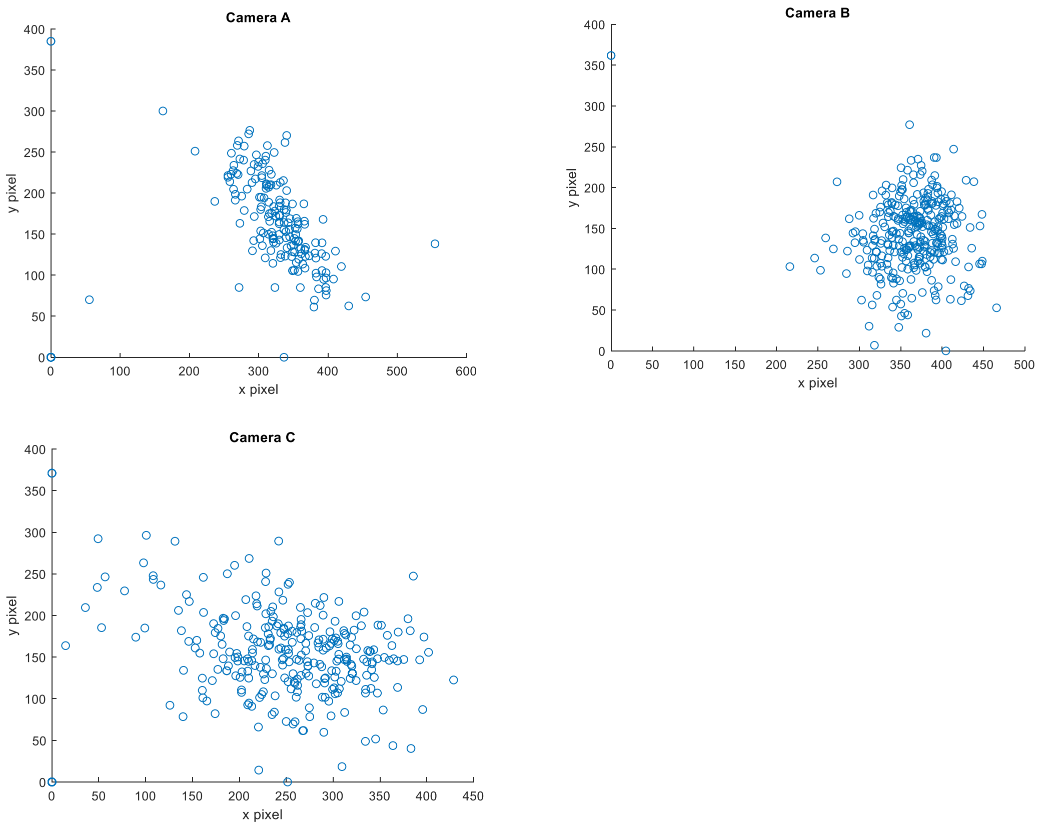
Sec. IV. Computational Results

For each of the four cases, I prepared scatter plots of the (x,y) values measured by each of the cameras. The ideal case showed very clear motion along a single component for cameras a and b, with a little bit more outliers in the motion of camera c. The noisy case showed a significantly wider spread along a secondary direction, which is expected with a significant amount of camera shake. In the horizontal motion case, there seemed to be stronger alignment along a single component than the noisy case. In the horizontal motion and rotational motion case, there seemed to be stronger alignment along a single component than both the noisy case and the horizontal motion case.

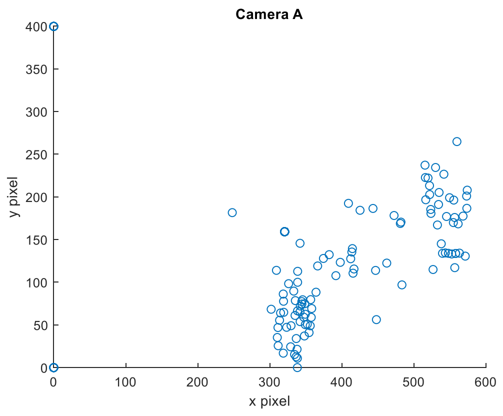
[1] Kutz, J. Nathan. Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data, Oxford University Press, 2018

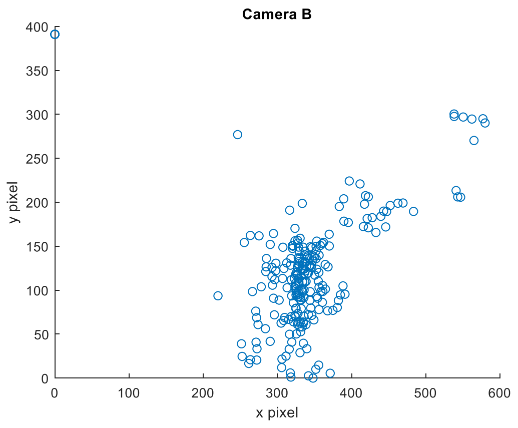


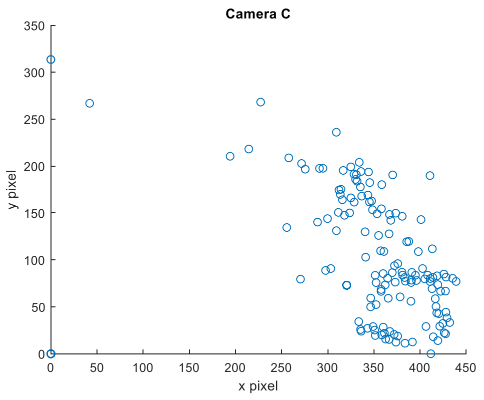
Figures 1 a), b), c), and d) – Individual camera measurements of the ideal case



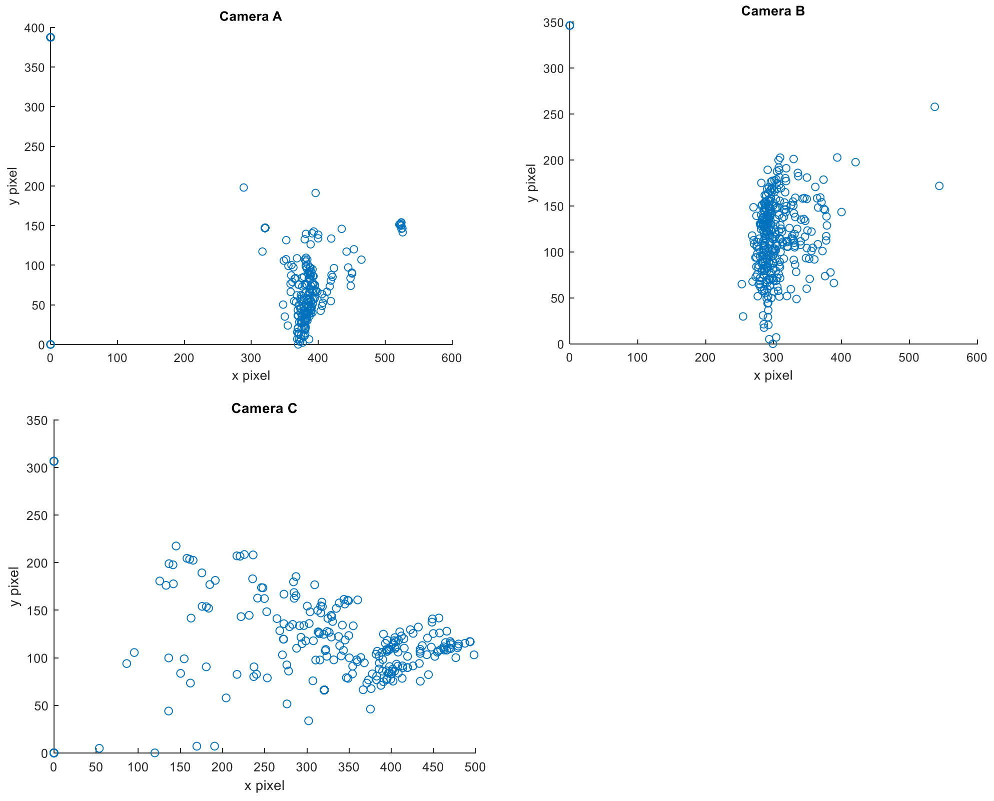
Figures 2 a), b), c), and d) – Individual camera measurements of the noisy case



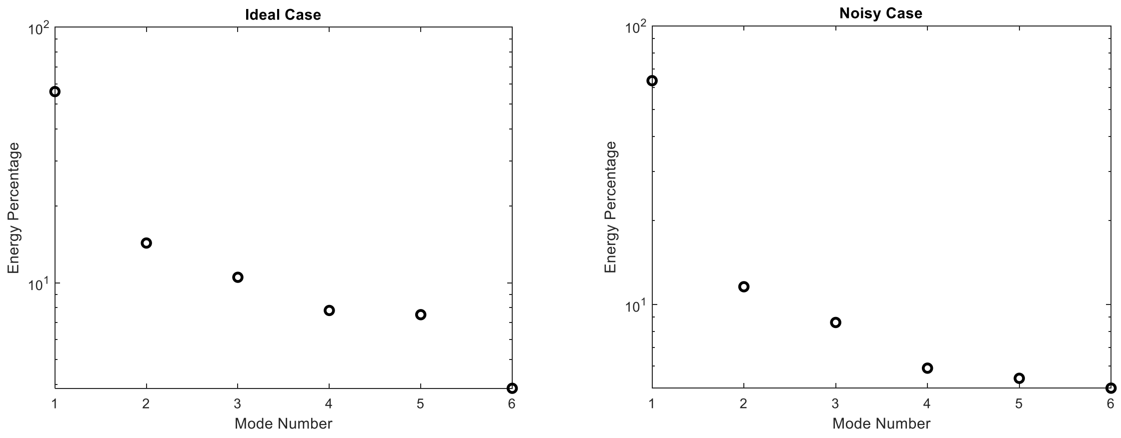


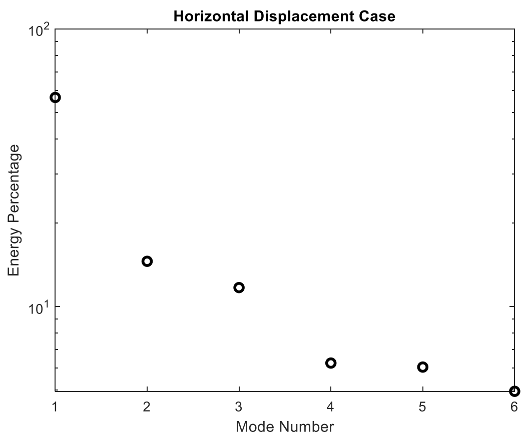


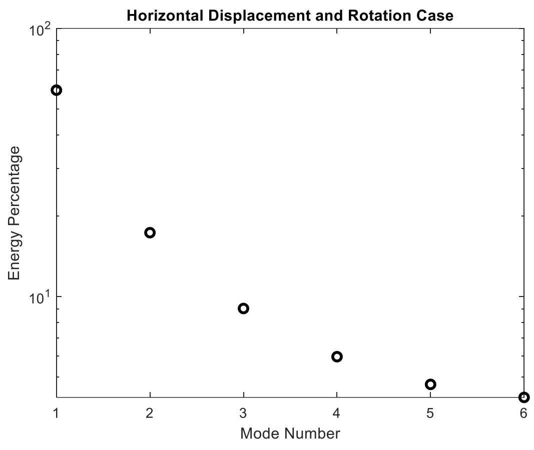
Figures 3 a), b), c), and d) – Individual camera measurements of the horizontal motion case



Figures 4 a), b), c), and d) – Individual camera measurements of the horizontal and rotational motion case







Figures 5 a), b), c), and d) – Principal components of each of the four cases as percentages of total energy

For the ideal case, the first three modes had respective energies of 56.0%, 14.3%, and 10.5%, for a total of 80.8% of the total energy.

For the noisy case, the first three modes had respective energies of 63.5%, 11.6%, and 8.6%, for a total of 83.7% of the total energy.

For the horizontal motion case, the first three modes had respective energies of 56.5%, 14.5%, and 11.7%, for a total of 82.6% of the total energy.

For the horizontal motion and rotational motion case, the first three modes had respective energies of 58.8%, 17.3%, and 9.02%, for a total of 85.1% of the total energy.

Sec. V. Summary and Conclusions

I was able to extract motion of the paint cans for each of cameras in each of the four cases. From the motion of the paint can measured by each camera, I performed SVD to determine the principal components. For each of the four cases, the first three modes of the principal components represented over 80% of the total energy.

Appendix A MATLAB functions used and brief implementation explanations

bwareaopen – This removes objects with fewer than a specified number of pixels from a binary image. My algorithm used 200 pixels in order to isolate the paint can.

bwlabel – This labels connected components in a binary image. My algorithm used this to label the centroid of the paint can.

cellfun() – This applies a specified function to each cell in an array.

evalin() – I used this to extract the current video data from the loaded MATLAB data in the base workspace.

graythresh() – This returns a threshold to use to convert a grayscale image to binary.

imabsdiff() – This returns the absolute different between two images.

max() – This returns the maximum value in an array.

padarray() – This is used to pad zeroes to make the arrays corresponding to each camera data the same size after synchronizing the video.

regionprops() – This returns properties of a region of pixels. I used it to return the centroid of a connected region.

rgb2gray() – This converts a color image to a grayscale image.

size() – This returns the size of an array.

svd() – This calculates the three matrices of SVD based on the input matrix.

vertcat() – This vertically concatenates arrays. I used it to combine the vectors of the centroid of the paint can measured by each camera into a final array to be used for SVD.

Appendix B MATLAB codes

```
clear all; close all; clc

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

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

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

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

```matlab
videos = {'vidFrames1_1', 'vidFrames2_1', 'vidFrames3_1'};
%videos = {'vidFrames1_2', 'vidFrames2_2', 'vidFrames3_2'};
%videos = {'vidFrames1_3', 'vidFrames2_3', 'vidFrames3_3'};
%videos = {'vidFrames1_4', 'vidFrames2_4', 'vidFrames3_4'};

matrix=cell(3,1);
for cameraNumber=1:3
    currentVideo = videos{cameraNumber};
    vidFrames = evalin('base',currentVideo);
    numFrames = size(vidFrames, 4);
    for frameNumber = numFrames:-1:1
        mov(frameNumber).cdata = vidFrames(:,:,:,frameNumber);
        mov(frameNumber).colormap = [];
    end
    clear diff
    XY_coord = zeros(1, numFrames-1);
    y_max=0; %track lowest value to synchronize videos
    for frameNumber=numFrames:-1:2
        currentFrame = rgb2gray(mov(frameNumber).cdata);
        previousFrame = rgb2gray(mov(frameNumber-1).cdata);
        difference(:,:,1) = imabsdiff(currentFrame, previousFrame);
        threshhold = graythresh(difference)*2;
        bw = (difference >= threshhold * 255);
        bw2 = bwareaopen(bw, 200); %remove all components that have fewer 
than 200 pixels
        s = regionprops(bwlabel(bw2(:,:,1)), 'centroid');
        x_avg=0;
        y_avg=0;
        count = 0;
        for j=1:numel(s)
            if s(j).Centroid(1) > 580
                continue
            end
            x_avg=s(j).Centroid(1)+x_avg;
            y_avg=s(j).Centroid(2)+y_avg;
            count = count + 1;
        end
        x_avg=x_avg/(count);
        y_avg=y_avg/(count);
        XY_coord(1, frameNumber-1) = x_avg;
        XY_coord(2, frameNumber-1) = y_avg;
        if y_avg > y_max
            y_max = y_avg;
        end
    end
    XY_coord(isnan(XY_coord)) = 0;
    XY_coord(2,:) = y_max - XY_coord(2,:);
    matrix{cameraNumber} = XY_coord;
end

% build final matrix from each vector

longest = cellfun('length', matrix);
longest = max(longest);
X = zeros(6, longest);
```

```
for n=1:3 %pad zeroes where needed
    temp_matrix = matrix{n,1};
    if (numel(temp_matrix(1,:)) < longest)
        temp_matrix = padarray(temp_matrix, [0 longest -
numel(temp_matrix(1,:))], 'post');
    end
    matrix{n,1} = temp_matrix;
end

X = vertcat(matrix{1,1}, matrix{2,1}, matrix{3,1}); %The final matrix

[u,s,v]=svd(X); % perform SVD on the final matrix

semilogy(100*diag(s)/sum(diag(s)), 'ko','Linewidth',2) %plot the diagonals of
s
xlabel('Mode Number')
ylabel('Energy Percentage')
title('Horizontal Displacement and Rotation Case')
xticks([0:1:6])

%scatter(X(1,:),X(2,:))
%scatter(X(3,:),X(4,:))
%scatter(X(5,:),X(6,:))
%title('Camera A')
%title('Camera B')
%title('Camera C')
%xlabel('x pixel')
%ylabel('y pixel')
```