Principal Component Analysis on Simple Harmonic Motion of a Paint Can

Ofek Inbar

Abstract

We explored the Principal Component Analysis of the motion of a paint can moving in simple harmonic motion.

1 Introduction and Overview

We're given a number of recordings of a paint can moving in simple harmonic motion (in the y direction). After tracking the paint can's x and y positions, we broke down the signal's variation over time into its Principal Components, so as to see the effect of different camera angles or of a noisy signal (shaky camera) on the produced result.

2 Theoretical Background

Principal Component Analysis is a method that enables us to take a multi-dimensional signal and break it down into a set of orthogonal "modes" and corresponding scalar values (the strength of the mode in the signal) onto which we can project the original data to get an idea for the internal "axes" it abides by.

The first step (after normalizing the data) is to apply Singular Value Decomposition to our signal data. This produces a square matrix U corresponding to the various "modes" found in the signal, a digonal matrix S whose diagonal

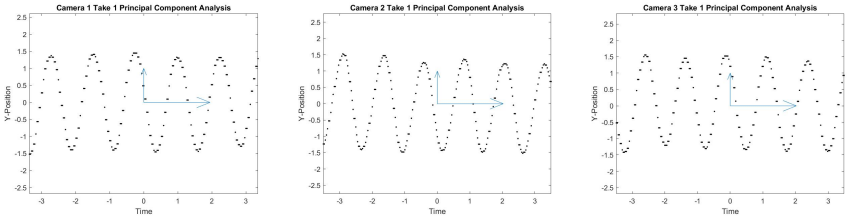

Figure 1: Y-Positions of the paint can in the "Ideal case"

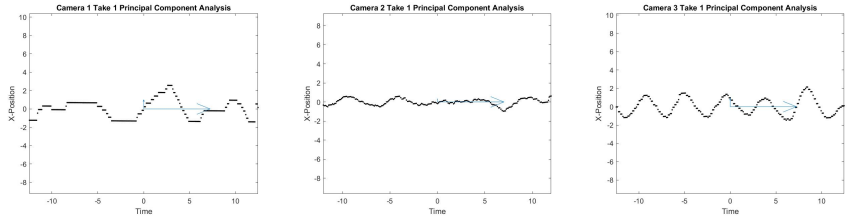

Figure 2: X-Positions of the paint can in the "Ideal case"

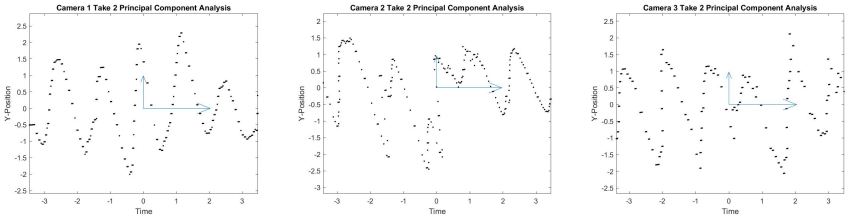

Figure 3: Y-Positions of the paint can in the "Noisy case"

elements are scalars for the respective strength of the "modes" in U, and a matrix V whose dimensions match the transposed dimensions of the original signal data.

3 Algorithm Implementation and Development

The algorithm for tracking the paint can is:

1. Determine the center of the paint can on the first frame (via a user click).
2. Store the RGB intensities of the square centered at the point found above with a side-length approximately equal to that of the can.
3. Move forward 1 frame.
4. For each pixel in the bounding box surrounding the can's center from the previous frame, compare the RGB intensities of a square of the correct size

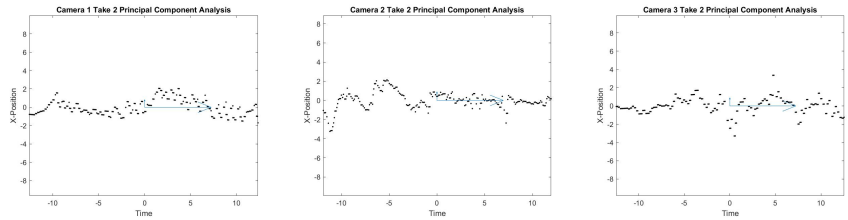

Figure 4: X-Positions of the paint can in the "Noisy case"

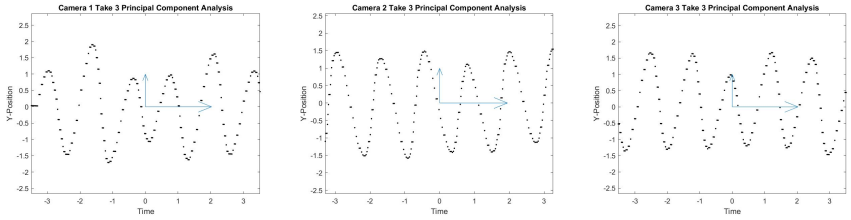

Figure 5: Y-Positions of the paint can with "Horizontal displacement"

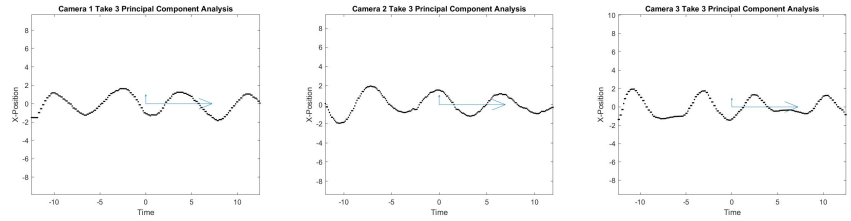

Figure 6: X-Positions of the paint can with "Horizontal displacement"

centered at this pixel with the intensities of the square from the previous frame. The new center of the paint can is the pixel for which the error (sum of squared differences in intensities) was smallest.

5. Go to step 3.

This was done for each test case of each camera, and the results (the coordinates of the center of the paint can for each frame) were saved.

After that, it was simply a matter of performing the standard PCA algorithms on the correlation between time on 1 axis and the resulting x and y values.

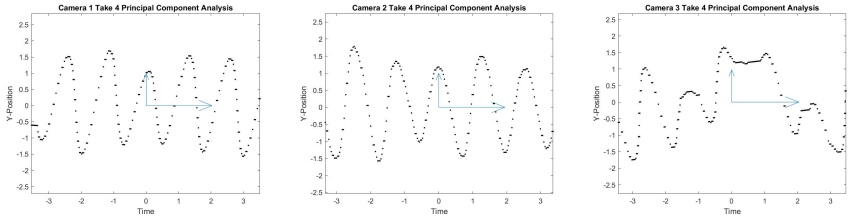

Figure 7: Y-Positions of the paint can with "Horizontal displacement and rotation"

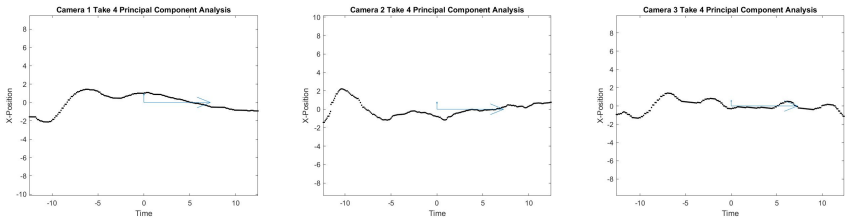

Figure 8: X-Positions of the paint can with "Horizontal displacement and rotation"

4 Computational Results

The clearest representation of the simple harmonic motion of the paint can can be seen in Figure 1. This figure shows the plots for the y-position of the paint can in cameras 1, 2, and 3 (respectively) in the ideal case, projected onto their Principal Component "modes". Note that the axes for the plots are very close to each other, and the same is true for the relative magnitudes of the Principal Component vectors. This is what we expect, given that all of the cameras were able to capture the entirety of the can's vertical motion. The plots of the x-positions (see Figure 2) shows that the cameras definitely had different perspectives of the can, as the horizontal displacement is very different across cameras. Note, still, that the relative magnitudes of the Principal Component vectors is the same.

In Figure 3 we see what happens when a "camera shake" is added. The vertical motion of the can still appears periodic, although its peaks and troughs occur at different heights for each time step, since the camera is always moving. This did not hinder our ability to extract the Principal Component vectors from the data and those appear constant across camera angles, and even appear to be pretty close to those of the y-positions from the "ideal case". Figure 4 shows the x-positions for the same scenario. The only interesting thing about these is there is no longer any discernible periodic motion in the plot for Camera 3, as there was in the first scenario.

Figures 5 and 6 show the x- and y-positions (respectively) of the paint can in the third scenario, where a horizontal displacement was added along with the usual vertical motion. It is clear to see from the plots of the x-positions that there is simple harmonic motion present, in a way that wasn't for Cameras 1 and 2 in the first scenario.

Finally, Figures 7 and 8 show the plots for x- and y-positions of the paint can in the fourth scenario, wherein rotation was added along with the horizontal displacement from the third scenario. For some reason, the we were unable to get the x-position to track very well on any camera for this case, and even the y-position got messed up for a period of time for the third camera. We tried a variety of different sizes for the size of the box to track and the original center, but were never able to fix these issues. Unfortunately, this means we aren't able to clearly see the harmonic motion in x at all. It is worth noting that the PCA

algorithms still produced Principal Components that are reflective of the data and relatively consistent across different cameras.

5 Summary and Conclusions

Principal Component Analysis allows us to break down a set of correlated data points into a number of orthogonal modes (which can sometimes be thought of as axes or bases), and they can tell us about the spread of the data along the directions of these modes. Camera shake does not appear to hinder these algorithms from identifying the modes and their magnitudes.

6 Appendix A

We used `imshow` to display the camera's RGB data frame-by-frame to allow us to track the can with our eyes, and `ginput` to record the position of a mouse click on the image.

7 Appendix B

The `MATLAB` code for producing the figures shown is below:

```
%% Part 1 - Track Paint Can
clear variables; close all; clc;

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

box_size = 40;
search_size = 40;

centers1 = trackPoint(vidFrames1_1, box_size, true);
centers2 = trackPoint(vidFrames2_1, box_size, true);
centers3 = trackPoint(vidFrames3_1, box_size, true);

save('centers1_1', 'centers1');
save('centers2_1', 'centers2');
save('centers3_1', 'centers3');

%% Part 2 - PCA
clear variables; close all; clc;

load('centers1_1.mat');
load('centers2_1.mat');
```

```
load('centers3_1.mat');

data = centers3(:, 2:-1:1); % camera is sideways
%data = centers2;
t = 1:226;
X = [t; normalize(double(data(t, 1)))'];
X(1, :) = X(1, :) - mean(X(1, :));
X(1, :) = X(1, :) * 25 / t(end);
%X(2, :) = X(2, :) - mean(X(2, :));

[U,S,V] = svd(X, 'econ');
n = size(V, 1);
y1 = S(1,1)/sqrt(n-1)*U(:,1);
y2 = S(2,2)/sqrt(n-1)*U(:,2);

X = U'*X;
y1 = U'*y1;
y2 = U'*y2;

plot(X(1, :), X(2, :), 'k.');
hold on;
c = compass(y1(1),y1(2));
c = compass(y2(1),y2(2));
axis equal;
xlabel('Time');
ylabel('X-Position');
title('Camera 3 Take 1 Principal Component Analysis');

saveas(gcf, 'centers3_1_x.jpg');
```

The code above relies on the following two MATLAB functions that we implemented for this assignment:

```
function [centers] = trackPoint(data, box_size, showBox)
    search_size = box_size;
    numFrames = size(data,4);

    imshow(data(:, :, :, 1));
    point = int16(ginput(1));
    close all;

    y = point(2) + int16(-box_size:box_size);
    x = point(1) + int16(-box_size:box_size);
    tracking_box = data(y, x, :, 1);

    centers = [point];
```

```
    for j = 2:numFrames
        X = (data(:,:,:,j));

        % find new center

        point = findCenter(X, tracking_box, point, box_size, search_size);
        centers = [centers; point];

        if showBox
            % draw green box
            for x = (point(1) + int16(-box_size:box_size))
                for y = (point(2) + int16([-box_size, box_size]))
                    X(y, x, 1) = 0;
                    X(y, x, 2) = 255;
                    X(y, x, 3) = 0;
                end
            end
            for y = point(2) + int16(-box_size:box_size)
                for x = point(1) + int16([-box_size, box_size])
                    X(y, x, 1) = 0;
                    X(y, x, 2) = 255;
                    X(y, x, 3) = 0;
                end
            end

            tracking_box = X(point(2) + int16(-box_size:box_size), ...
                point(1) + int16(-box_size:box_size), :);

            imshow(X);
            drawnow
        end
    end
end

function [minPoint] = findCenter(X, tracking_box, point, box_size, search_size)
    min_error = (256*2*box_size)^2;

    min_x = max(box_size + 1, point(1) - search_size);
    max_x = min(size(X, 2) - box_size, point(1) + search_size);

    min_y = max(box_size + 1, point(2) - search_size);
    max_y = min(size(X, 1) - box_size, point(2) + search_size);

    for x = min_x:max_x
        for y = min_y:max_y
            a = X(y + int16(-box_size:box_size), x + int16(-box_size:box_size), :);
```

```
            error = (int16(tracking_box) - int16(a)).^2;
            tmp_error = sum(error, 'all');
            if tmp_error < min_error
                min_error = tmp_error;
                min_point = [x, y];
            end
        end
    end

    minPoint = min_point;
end
```