Analyzing Paint Can Motion with Principal Components Analysis

Author: Kuan-Wei Lee

Abstract:

The goal of this project is to analyze paint can motion captured by three different cameras with the principal components analysis. The strategy of the analysis is tracking the paint can motion with vision tool box in MATLAB, finding principal components and performing dimensionality reduction with singular values decomposition (SVD).

Sec I.

Introduction

In this project, I am analyzing several videos that captured the motions of a paint can by different camera from different angles. An important first step of this project is to record the position of the paint can in a given time, which means that I need to develop an algorithm that tracks the position of the paint can and return the x-y position of the paint can in a given frame. I accomplished the task by using the vision toolbox in MATLAB. After tracking the position of the paint can, I got a matrix X that contains the x-y position of the paint can captured by the three cameras from each set of videos. The challenge on modeling the motion of the paint can with the matrix X is that the matrix may contains some noises and redundant information. For example, the camera may be shaking while taking the videos and the motion of the paint can could be along one dimension so that only the information from one of the camera is necessary to model the paint can motion. To eliminate most of the noise and redundant information, I am going to rely on the principal component analysis and the singular values decomposition.

Sec II.

Theoretical Background

Singular values decomposition and principal components analysis:

1. Singular values decomposition (SVD):

Singular values decomposition is a factorization of a matrix under a number of constitutive components. We can also see the SVD as a transformation that stretches/compress a set of vectors. Consider a $m \times n$ matrix A that stretches an n dimensional hypersphere in $\mathbb{R}^n$ to a hyperellipse in $\mathbb{R}^m$. A hyperellipse in $\mathbb{R}^m$ is defined as by the surface obtained by stretching a unit sphere in $\mathbb{R}^m$ by some factors $\sigma_1, \sigma_2, \dots, \sigma_m$ in the orthogonal directions $u_1, u_2, \dots, u_m$. The transformation from the unit sphere to the hyperellipse can be expressed as follows:

$$A\, v_j = \sigma_j\, u_j \quad 1 \le j \le n \qquad (1)$$

In a more compact matrix notation, the expression above becomes:

$$A\,V = \hat{U}\,\hat{\Sigma} \qquad (2)$$

The matrix $\hat{\Sigma}$ is an $n \times n$ diagonal matrix with positive entries σ_j where the values of σ_j is ordered from the largest to the smallest. The matrix $\hat{U}$ is an $m \times n$ matrix with orthogonal columns. The matrix V is an $n \times n$ unitary matrix. Since V is an unitary matrix, the matrix A can be solved by multiplying the above equation by the transpose of matrix V.

$$A = \hat{U}\,\hat{\Sigma}\,V^* \qquad (3)$$

The factorization is known as the reduced SVD of the matrix A.

2. Principal components analysis

One common application of SVD is principal components analysis. In the previous section I mentioned that SVD factorizes a matrix to $\hat{U}, \hat{\Sigma},\ V^*$. The rows of the matrix $\hat{U}$ are the principal components. The principal components are basically the vectors that shows the direction where the original matrix vary. The principal components correspond to the highest singular value is the direction where the original data/matrix vary the most. By examining the singular values of the SVD, we can see the relative importance of each principal components and understand the overall dynamic of the data. If we approximate the original data with a few important modes that capture most of the dynamic, we can achieve data compression and dimensionality reduction, which helps us to reduce the size of the data and remove redundancy.

Sec III.

Algorithm Implementation and Development

As I mentioned in the previous section, the objective of this project is to analyze the motions of the paint can in the videos. To analyze the motion of the paint can, we need to obtain the coordinates of the paint can in every frame. The strategy I use to obtain the coordinates is using the vision tool box in MATLAB. I developed an object tracking algorithm that track a region of points on the paint can in the video and obtained the position of the paint can by averaging the points being tracked. I stored the positions of the paint can obtained from three different cameras into a single matrix. Since the videos taken by the threes cameras have different number of frames, I resize the data from the three cameras to match the dimension. I set the first frame where the paint can is at its maximum height and set the end of the frame to the 225th frame after the first frame. I consistently use the above strategy throughout my analysis of the four sets of videos. After I obtained the coordinates of the paint can and stored them into the matrix ‘X’, I use the MATLAB function ‘svd’ to perform principal components analysis. I plotted the diagonals of the singular values matrix to see how many modes are dominant. I also plotted the temporal modes ‘V’ to see how the motion changes with time.

Sec IV.

Computational Results

When I perform PCA on the first set of video by performing SVD on the matrix X, I found that the first singular value is much greater than the rest of the singular values (Figure 1), which suggests that the first mode is the dominant mode. In the context of motion detection, the first mode dominate means that most of the data from the three cameras are redundant. A way to interpret this is that the motion captured by the camera is one dimensional. When I plotted the temporal modes (matrix V), I found that the first three temporal modes oscillate like sin waves while the fourth mode wiggles like a noise (Figure 2). The first three oscillating temporal modes suggest that the motion captured by the individual cameras is some kind of oscillation and the fourth temporal mode could be given by the noise such as the shaking of the camera. The finding is consistent with the actual video since we can see that the paint can oscillating up and down in the three videos.

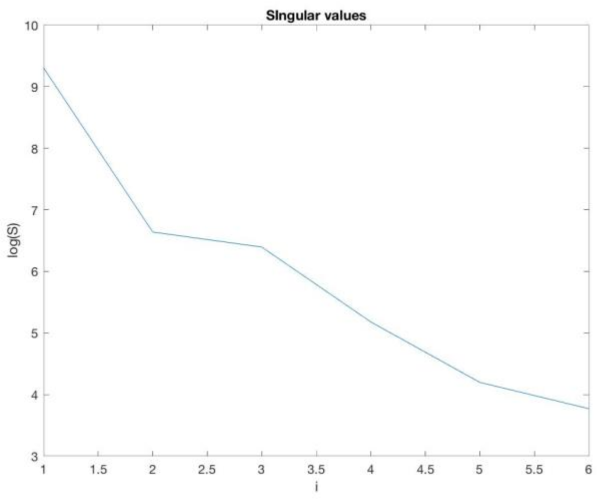

Figure 1. Singular values of the matrix X. It is plotted in the log scale and the graph shows that the first singular values is much bigger than other singular values.

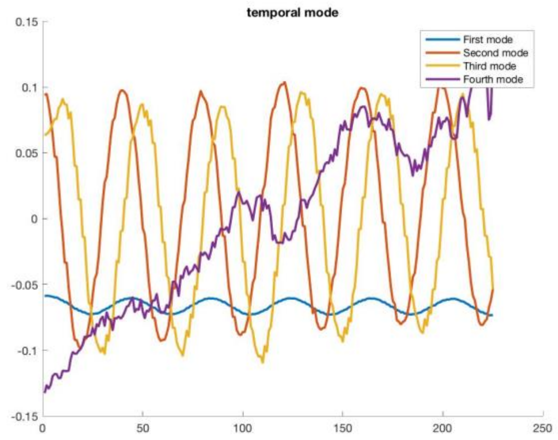

Figure 2. First four temporal modes of matrix X.

In the second set of videos, the cameras are still capturing the oscillation of the paint can. The only difference is that the three cameras are shaken intentionally to create some noise. When I plotted the singular values of the matrix X2 (position of the paint can in second set of videos) like I did for the first set of videos, the first singular is still much bigger and the first mode still dominate (Figure 3). When I graph the first four temporal modes of X2, I still can see the oscillation in the first three modes but the sin waves look a lot more noisy compared to the temporal modes of X1 (Figure 4).

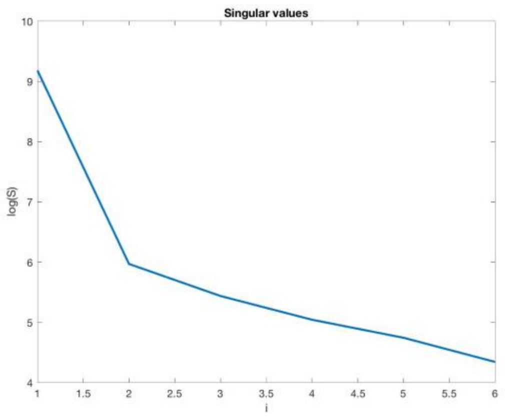

Figure 3. Singular values of matrix X2.

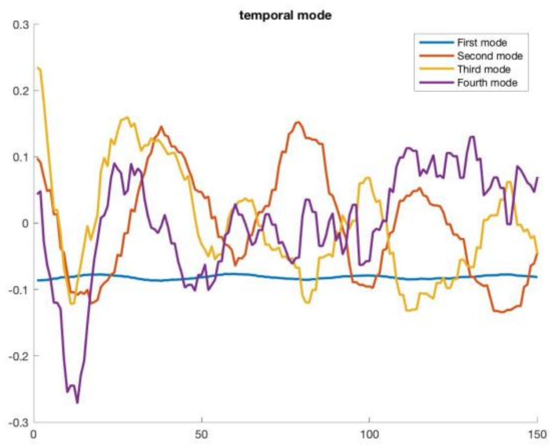

Figure 4. Temporal modes of X2.

The Singular values of the matrix X3(position of the paint can in the third set of videos) have the similar trend to the singular values of X1 and X2. The first singular value is much greater than the rest. The interesting difference in the SVD of X3 is that the second temporal mode oscillates like a sin wave but the amplitudes of the oscillation changes (Figure 5). The change of oscillation amplitude of the second temporal modes may suggest that there is one extra degree of oscillation captured by at least one out of the three cameras.

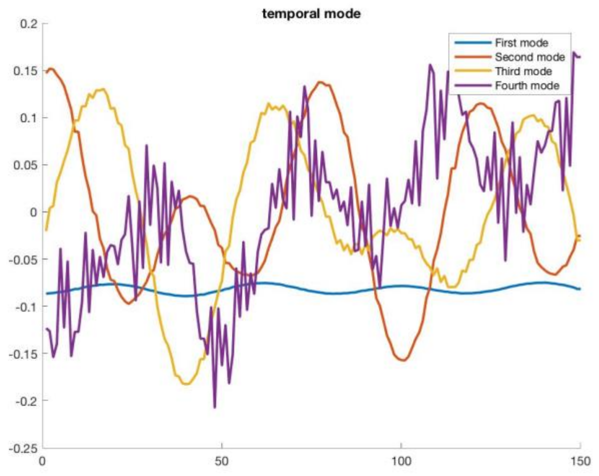

Figure 5. Temporal modes of X3.

The fourth set of video is the oscillation of the paint can plus the rotation of the paint can. Tracking motions like this is very tricky for the algorithm I chose because the point trackers lost track of the points when the points being track rotate to the other side and disappear from the line of sight of the cameras. With my current knowledge, I cannot think of a good strategy to track the points.

Sec V.

Conclusion and Summary

SVD is a very powerful tool to perform PCA. By including only the most important modes, we can remove most of the redundant information and capture most of the dynamics. In this project, we can reconstruct and analyze the motion with the principal components. If we perform dimensionality reduction, we could even remove the noise created by the wobbling of the cameras.

Appendix A.

```
function [position] = trackobj(objectRegion,frames)

v = VideoWriter('motion.avi');
open(v)
writeVideo(v,frames)
close(v)
videoFileReader = vision.VideoFileReader('motion.avi');
videoPlayer = vision.VideoPlayer('Position',[100,100,680,520]);
objectFrame = videoFileReader();

objectImage =
insertShape(objectFrame,'Rectangle',objectRegion,'Color','red');
figure;
imshow(objectImage);
title('Red box shows object region');

% points =
detectMinEigenFeatures(rgb2gray(objectFrame),'ROI',objectRegion);
x = objectRegion(1):1:objectRegion(1)+objectRegion(3);
y = objectRegion(2):1:objectRegion(2)+objectRegion(4);
[X,Y] = meshgrid(x,y);
points(1,:) = X(:);
points(2,:) = Y(:);
pointImage = insertMarker(objectFrame,points','+','Color','blue');
figure;
imshow(pointImage);
title('Detected interest points');
tracker = vision.PointTracker('MaxBidirectionalError',6);
blkmatcher = vision.BlockMatcher();
initialize(tracker,points',objectFrame);
k = 1;
```

```
while ~isDone(videoFileReader)
      frame = videoFileReader();
      [points,validity] = tracker(frame);
      posi = points;
      position(1,k) = mean(posi(:,1));
      position(2,k) = mean(posi(:,2));
      out = insertMarker(frame,points(validity, :),'+');
      videoPlayer(out);
      pause(0.1)
      k = k+1;
end

end
```

The above function is the algorithm that implement vision tool box to track the points on the paint can. The two inputs this function takes are the bounding box that encircle the points we want to track and the video frames. It return the averaged coordinates of the points tracked in each frame.

```
[U1,S1,V1] = svd(X1,'econ');
```

The above function is the SVD. It performs singular values decomposition to the matrix X.

Appendix B.

```
clear all
close all
clc

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

% implay(vidFrames1_1)

I = rgb2gray(vidFrames1_1(:,:,:,1));

objectRegion = [315,215,20,20];
objectRegion2 = [265,285,20,20];
objectRegion3 = [330,280,20,20];

position1 = trackobj(objectRegion,vidFrames1_1);
position2 = trackobj(objectRegion2,vidFrames2_1);
position3 = trackobj(objectRegion3,vidFrames3_1);
%%
l = length(position1(1,:));
start2 = 20;
```

```
start3 = 6;

X1 = [position1(1,:);position1(2,:);...
    position2(1,start2:start2+l-1);position2(2,start2:start2+l-1);...
    position3(1,start3:start3+l-1);position3(2,start3:start3+l-1)];

[U1,S1,V1] = svd(X1,'econ');

plot(log(diag(S1)))
title('SIngular values')
ylabel('log(S)')
xlabel('i')

figure(2)
hold on
x = 1:1:length(V1(:,1));
plot(x,V1(:,1),x,V1(:,2),x,V1(:,3),x,V1(:,4),'LineWidth',2)
legend('First mode','Second mode','Third mode','Fourth mode')

title('temporal mode')


%% Shaking camera
clear all
close all
clc

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

% implay(vidFrames1_2)

start1 = 15;
start2 = 1;
start3 = 19;
objectRegion1 = [325,315,30,30];
objectRegion2 = [280,325,50,50];
objectRegion3 = [370,250,30,30];

position1 = trackobj(objectRegion1,vidFrames1_2);
position2 = trackobj(objectRegion2,vidFrames2_2(:,:,:,5:end));
position3 = trackobj(objectRegion3,vidFrames3_2);

%%
l = 150;
X2 = [position1(1,start1:start1+l-1);position1(2,start1:start1+l-
1);...
    position2(1,start2:start2+l-1);position2(2,start2:start2+l-1);...
    position3(1,start3:start3+l-1);position3(2,start3:start3+l-1)];
```

```
[U2,S2,V2] = svd(X2,'econ');

plot(log(diag(S2)),'LineWidth',2)
title('Singular values')
ylabel('log(S)')
xlabel('i')

figure(2)
hold on
x = 1:1:length(V2(:,1));
plot(x,V2(:,1),x,V2(:,2),x,V2(:,3),x,V2(:,4),'LineWidth',2)
legend('First mode','Second mode','Third mode','Fourth mode')

title('temporal mode')
%%
clear all
close all
clc

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

start1 = 19;
start2 = 46;
start3 = 10;

objectRegion1 = [325,300,30,30];
objectRegion2 = [260,320,20,20];
objectRegion3 = [360,220,20,20];

position1 = trackobj(objectRegion1,vidFrames1_3);
position2 = trackobj(objectRegion2,vidFrames2_3(:,:,:,1:end));
position3 = trackobj(objectRegion3,vidFrames3_3);

% implay(vidFrames1_3)
%%
start1 = 19;
start2 = 46;
start3 = 10;
l = 150;

X3 = [position1(1,start1:start1+l-1);position1(2,start1:start1+l-
1);...
    position2(1,start2:start2+l-1);position2(2,start2:start2+l-1);...
    position3(1,start3:start3+l-1);position3(2,start3:start3+l-1)];

[U3,S3,V3] = svd(X3,'econ');
```

```
plot(log(diag(S3)),'LineWidth',2)
title('Singular values')
ylabel('log(S)')
xlabel('i')

figure(2)
hold on
x = 1:1:length(V3(:,1));
plot(x,V3(:,1),x,V3(:,2),x,V3(:,3),x,V3(:,4),'LineWidth',2)
legend('First mode','Second mode','Third mode','Fourth mode')

title('temporal mode')

%%
clear all
close all
clc

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

start1 = 19;
start2 = 46;
start3 = 10;

objectRegion1 = [380,290,30,30];
objectRegion2 = [260,320,20,20];
objectRegion3 = [360,220,20,20];

position1 = trackobj(objectRegion1,vidFrames1_3);
position2 = trackobj(objectRegion2,vidFrames2_3(:,:,:,1:end));
position3 = trackobj(objectRegion3,vidFrames3_3);
%%
objectRegion2 = [260,270,30,30];

trackobj(objectRegion2,vidFrames2_4(:,:,:,1:end));


 %%
function [position] = trackobj(objectRegion,frames)

v = VideoWriter('motion.avi');
open(v)
writeVideo(v,frames)
close(v)
videoFileReader = vision.VideoFileReader('motion.avi');
videoPlayer = vision.VideoPlayer('Position',[100,100,680,520]);
objectFrame = videoFileReader();
```

```matlab
objectImage = 
insertShape(objectFrame,'Rectangle',objectRegion,'Color','red');
figure;
imshow(objectImage);
title('Red box shows object region');

% points = 
detectMinEigenFeatures(rgb2gray(objectFrame),'ROI',objectRegion);
x = objectRegion(1):1:objectRegion(1)+objectRegion(3);
y = objectRegion(2):1:objectRegion(2)+objectRegion(4);
[X,Y] = meshgrid(x,y);
points(1,:) = X(:);
points(2,:) = Y(:);
pointImage = insertMarker(objectFrame,points','+','Color','blue');
figure;
imshow(pointImage);
title('Detected interest points');
tracker = vision.PointTracker('MaxBidirectionalError',6);
blkmatcher = vision.BlockMatcher();
initialize(tracker,points',objectFrame);
k = 1;
while ~isDone(videoFileReader)
      frame = videoFileReader();
      [points,validity] = tracker(frame);
      posi = points;
      position(1,k) = mean(posi(:,1));
      position(2,k) = mean(posi(:,2));
      out = insertMarker(frame,points(validity, :),'+');
      videoPlayer(out);
      pause(0.1)
      k = k+1;
end

end
```