AMATH 582 Homework 3

Pavel Zelinsky

Github: <https://github.com/palpalych/PaintCan>

**Abstract**

I am exploring the effects of the PCA method through SVD from the additions of problem dimensions and noise

## Introduction and Overview

I have data from 3 different cameras for a bucket doing harmonic oscillations in four different conditions. In the first recordings the buckets is moving up and down normally, in the second recording there is the introduction of noise through camera shaking, in the third recording the bucket is moving up and down as well as from side to side, and in the last recording the bucket is moving up and down as well as spinning. By applying the SVD to each of these scenarios I can investigate what are the primary components of every motion despite only having 2D images of what was going on

## Theoretical Background

The SVD is a matrix decomposition that breaks any matrix A into two unitary parts (U and V) and a diagonal part ∑

This can be formed by taking the eigenvectors and eigenvalues of AAT and ATA, where V is the eigenvectors of ATA, U is the eigenvectors of AAT, and ∑ is a diagonal matrix of the square roots of the eigenvalues, sorted from largest to smallest. The SVD has several particularly useful features for data analysis. First and foremost, every matrix has an SVD. This means that any additional benefits from the SVD can always be obtained no matter how ill-formed the data set. Second, it provides the most optimal rank K approximation by the L2 norm via this formula:

Lastly, if your data is put in as each column of A is a data point, then the first K columns of U\* can be used as a transform for your data into K principal components, with the rows of V cut off at the (K+1)th column are the projections of your data into the principal components:

Given that U and V are unitary, these components are orthogonal to each other. This helps split any data set into the most relevant parts. Note: if your data is instead inserted as rows of A, then everything is still true except U and V are reversed for which is the transform and which is the projections of A.

## Algorithm Implementation and Development

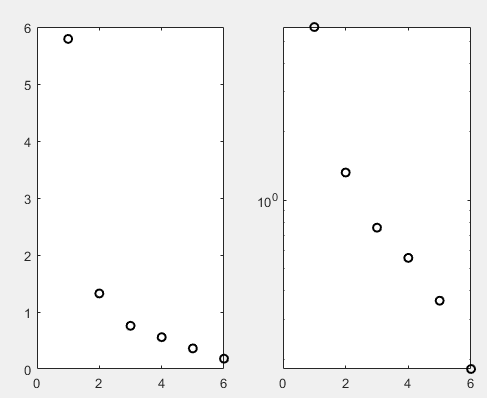
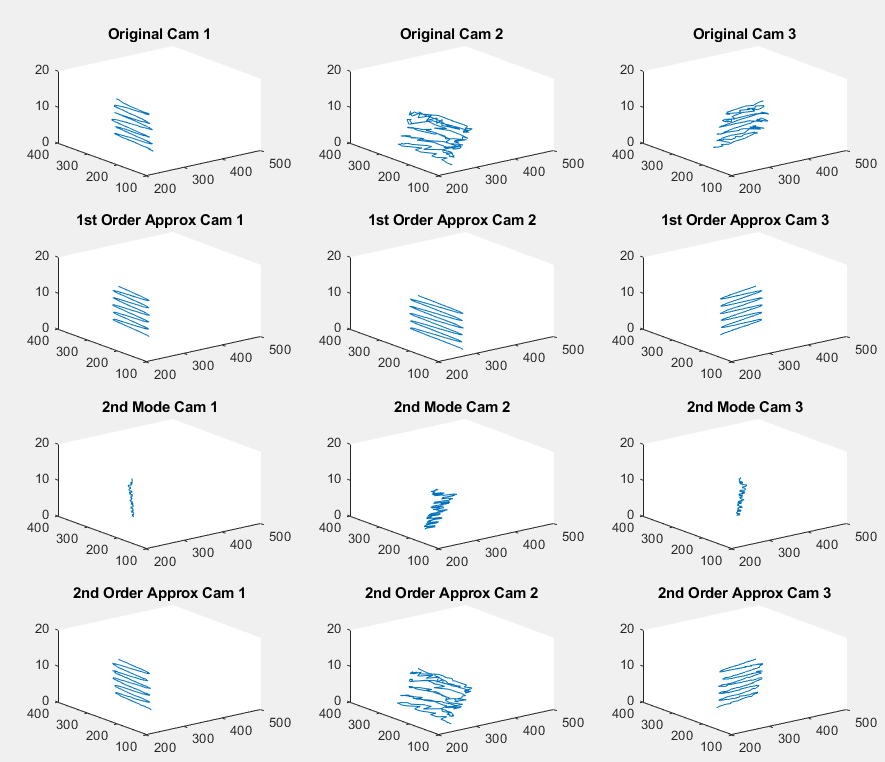
The most challenging part of this was tracking the paint can in the images without manual intervention. In order to help with this I first wrote a helper ExtractCommonColors. It collects a HashSet of all the colors that show up in the first frame of each movie, and then tracks whether each of those colors shows up within some error bounds in all other frames. At the end the helper gives me the set of colors that show up everywhere, allowing me to hone in on a particular color that is present on only the paint can and trace it. For that I wrote an additional helper, TraceColor, that in every frame tracks what top-leftmost x and y coordinates are in particular color range, and then proceeds to corrupt those frames and some nearby frames for easier visual inspection. For the actual SVD I used the Matlab built in svd functionality, and looked at the different approximations of the data matrices

Figure :Variance of Video 1

## Computational Results

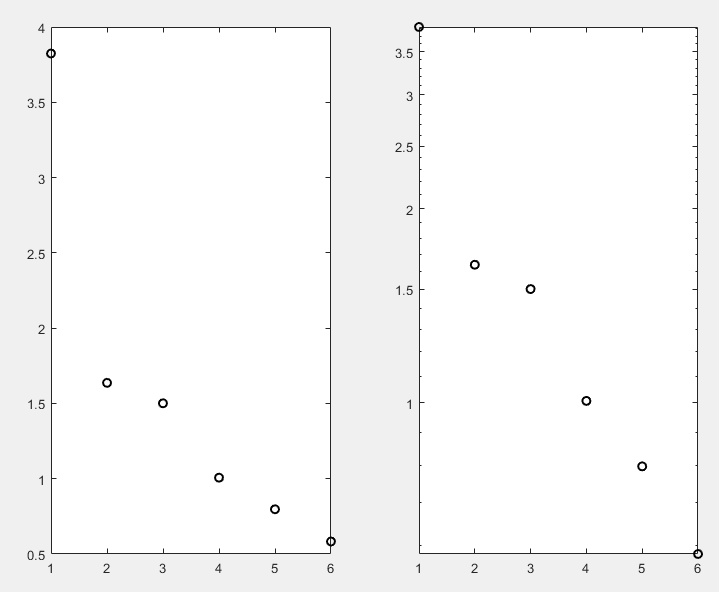
The first video was moving up and down only. So as expected, when looking at the variance of the principal components (Figure 1) there is clearly one that is dominant. This also shows in the low rank approximations (Figure 2) – the first order approximations gets the full motion of the oscillation and only misses some noise factors from the pixel being tracked on the bucket moving around slightly. The second mode seems to be gathering most of that noise, and so the second order approximation looks nearly identical to the original signal

Figure 3:Variance of Video 2

Figure :Video 1 low rank approximations

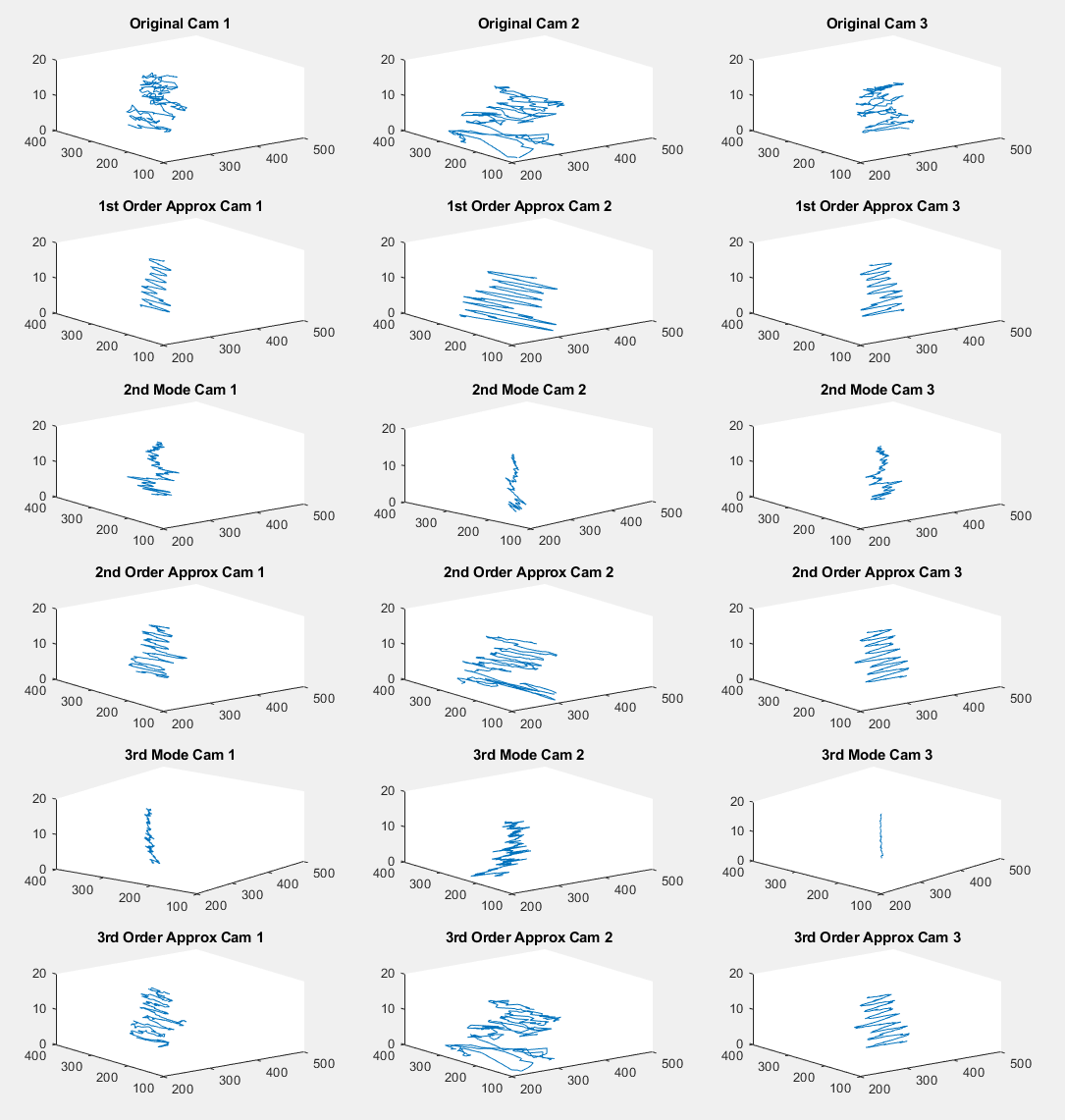
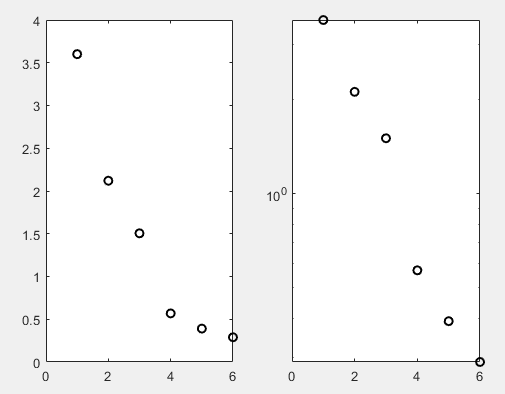
In the second video we have an added camera shake. From looking at the variances (Figure 3) this shows up as a heightened second and third mode. The first mode – the one for the up and down motion – is still the primary component, but the additional components from the noise are substantial enough to stand out. This is highlighted in the low rank approximations and modes (Figure 4), where even the original data is difficult to parse and for example in the original signal for Camera 1 it isn’t perfectly clear that there is an oscillation at all. The first order approximation does extract that main movement, but due to the noise even that component is uneven. As expected, the second and third modes are also relatively sizeable and up until the third order approximation we don’t quite get the same figure as the original. It is interesting to note that each of the modes is visibly pointed in slightly different directions, so each of them is showing a movement of the paint can in a distinct direction

Figure 4:Video 2 low rank approximations and modes

In the third video we have the paint can moving around as well as moving up and down. This means that we should expect to see 3 principal components. Looking at Figure 5 we get the 3 principal components but it should be noted that when comparing this to Figure 3 the second and third – especially the third – principal components are not substantially different. When looking at the actual modes and approximations (Figure 6), there is quite a different in how clean the additional modes are

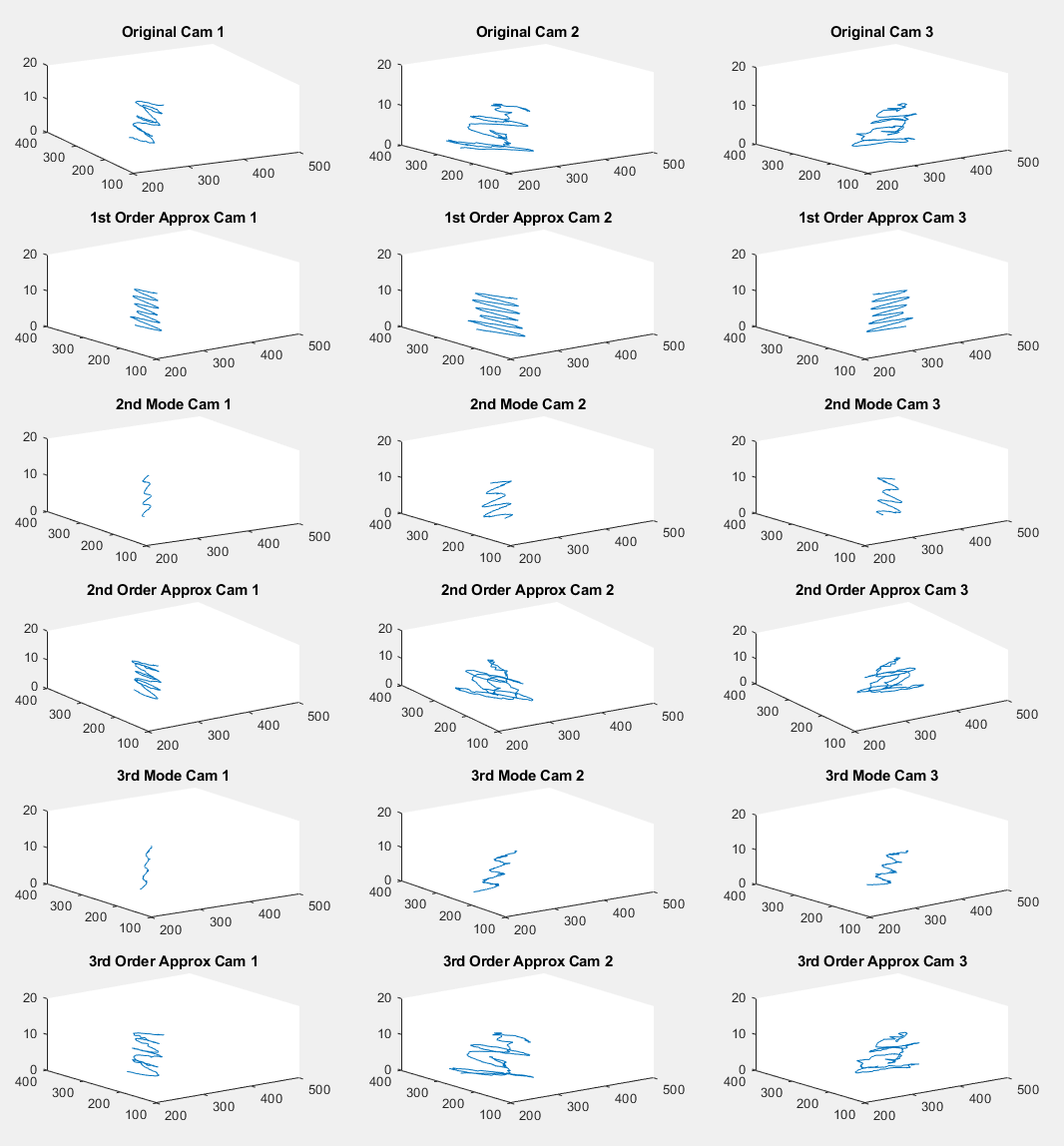
The final video has the paint can spinning as well as moving up and down. The spin should add some moving around motion similar to the third video, but when looking at Figure 8

Figure 5:Variance of Video 3

Figure 6:Video 3 low rank approximations and modes

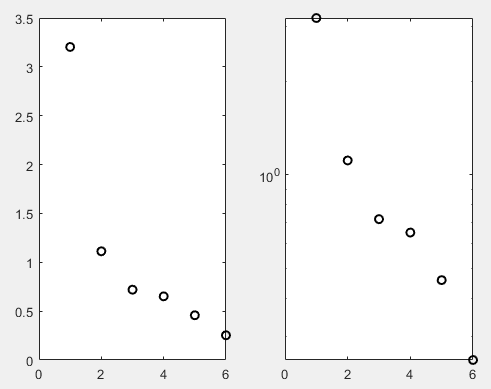
it isn’t perfectly evident that there is much more than the first principal component. It seems that although there was spin on the paint can, it didn’t actually cause much additional motion. This is shown in Figure 8 where even in the original camera images there is little motion outside of the up and down oscillations. In fact from both Figure 7 and Figure 8 we can see that the impact of the paint can rotating was smaller than the noise from shaking in Figures 3 and 4.

Figure 7:Variance of Video 4

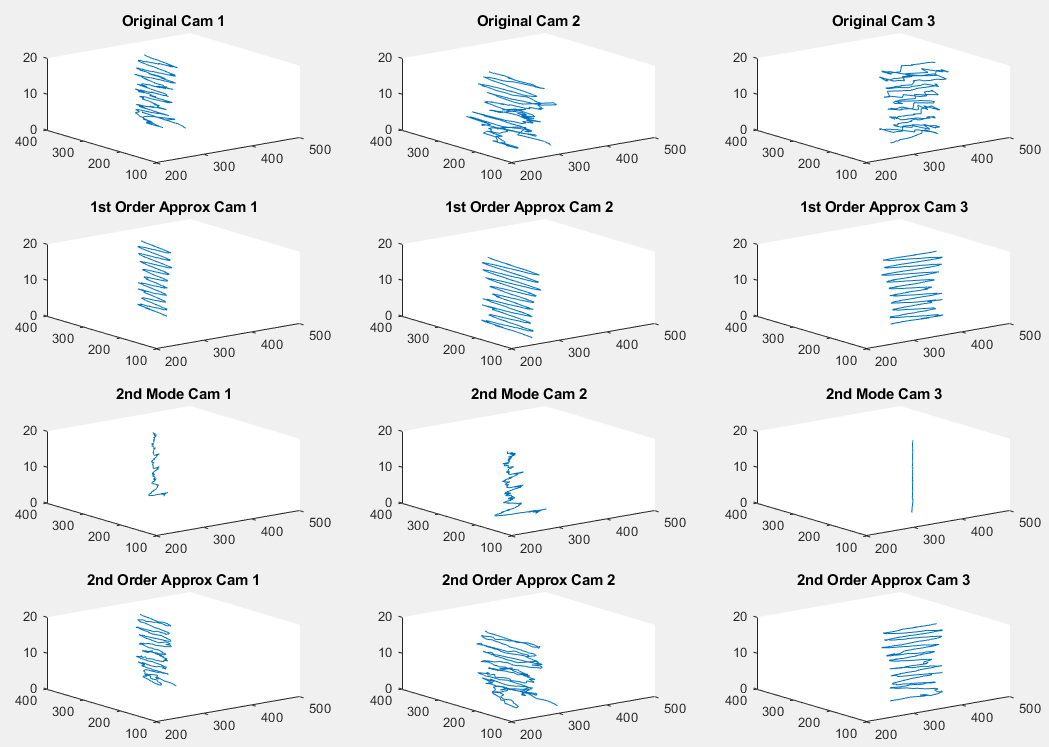


Figure 8:Video 4 low rank approximations and modes

## Summary and Conclusion

The SVD is useful in extracting the main components out of a system. Noise is able to have enough correlations within a system though to also have a substantial component

Appendix A – MATLAB Functions and Code Notes:

Semilogy – plot on a logarithmic scale

Pcolor, shading, set, xlabel, ylabel, title, colormap, figure, subplot – functions used for plotting, organizing, and labeling the data

Image, implay – functions for looking at images and playing videos. This was highly useful as Matlab allows looking at videos frame by frame, and even get the exact RGB values of specific frames

Notes on code:

Most of the time was spent on figuring out the paint can positions in the images. For some of the longer videos (Video 4) I had to manually fix some points because the color tracking broke down. Additionally in order to improve the color tracking I blanked out everything outside of the range of where the paint can swings around

Appendix B – MATLAB Code:

ExtractCommonColors.m

function [commonColorRanges] = ExtractCommonColors(movie)

%EXTRACTCOMMONCOLORS Return what colors within a +5 or -5 range are used

%across all frames in the movie

movie\_size = size(movie);

% Extract all the colors in the first frame

seenThisFrame = java.util.HashSet;

for j=1:movie\_size(1)

for k=1:movie\_size(2)

seenThisFrame.add(strcat(num2str(movie(j,k,1,1)),',',num2str(movie(j,k,2,1)),',',num2str(movie(j,k,3,1))));

end

end

seenFirstFrame = str2num(seenThisFrame.toArray());

% Go through the rest of the frames, we're doing this backwards because the

% last frame and the first frame tend to have the most differences, so

% doing so quickly reduces the number of colors we're checking against.

% Additionally only look at every 3rd frame since we expect adjacent frames

% to any particular frame to be similar

for index=-movie\_size(4):3:-2

i = -index;

toRemove = [];

sizeOfSeen = size(seenFirstFrame);

for j=1:sizeOfSeen(1)

matches = movie(:,:,1,i) < seenFirstFrame(j,1)+5 & movie(:,:,1,i) > seenFirstFrame(j,1)-5 & movie(:,:,2,i) < seenFirstFrame(j,2)+5 & movie(:,:,2,i) > seenFirstFrame(j,2)-5 & movie(:,:,3,i) < seenFirstFrame(j,3)+5 & movie(:,:,3,i) > seenFirstFrame(j,3)-5;

if max(max(matches)) == 0

toRemove = [toRemove j];

end

end

for j=-length(toRemove):-1

seenFirstFrame(-j,:) = [];

end

end

% clump the colors together

commonColorRanges = seenFirstFrame;

i = 1;

while true

toRemove = [];

sizeOfSeen = size(commonColorRanges);

if i+1 >= sizeOfSeen(1)

break

end

for j=i+1:sizeOfSeen(1)

if commonColorRanges(i,1) < commonColorRanges(j,1)+5 && commonColorRanges(i,1) > commonColorRanges(j,1)-5 && ...

commonColorRanges(i,2) < commonColorRanges(j,2)+5 && commonColorRanges(i,2) > commonColorRanges(j,2)-5 && ...

commonColorRanges(i,3) < commonColorRanges(j,3)+5 && commonColorRanges(i,3) > commonColorRanges(j,3)-5

toRemove = [toRemove j];

end

end

for j=-length(toRemove):-1

commonColorRanges(-j,:) = [];

end

i = i + 1;

end

commonColorRanges = sortrows(commonColorRanges);

end

TraceColor.m

function [x\_coords,y\_coords,corruptedVid] = TraceColor(video, minR, maxR, minG, maxG, minB, maxB)

%TRACECOLOR Traces top left hand point of a particular color in the video

movie\_size = size(video);

x\_coords = zeros(1,movie\_size(4));

y\_coords = zeros(1,movie\_size(4));

corruptedVid = video;

for i=1:movie\_size(4)

colorSpots = video(:,:,1,i) > minR & video(:,:,1,i) < maxR & video(:,:,2,i) > minG & video(:,:,2,i) < maxG & video(:,:,3,i) > minB & video(:,:,3,i) < maxB;

[y,x] = ind2sub(movie\_size(1:2), find(colorSpots, 1));

if isempty(x)

% "no color"

% i

continue

end

% note: if the color is not found an Error will happen here

x\_coords(i) = x;

y\_coords(i) = y;

% if x <= 1 || y <= 1

% "invalid coordinate"

% i

% continue

% end

% update a video copy for visual validation

corruptedVid(y,x,1,i) = 0;

corruptedVid(y+1,x,3,i) = 0;

corruptedVid(y-1,x,1,i) = 0;

corruptedVid(y,x-1,3,i) = 0;

corruptedVid(y+1,x-1,1,i) = 0;

corruptedVid(y-1,x-1,3,i) = 0;

corruptedVid(y,x+1,1,i) = 0;

corruptedVid(y+1,x+1,3,i) = 0;

corruptedVid(y-1,x+1,1,i) = 0;

end

end

PlotApproximations.m

function [] = PlotApproximations(A, u, s, v, mean\_A)

%PLOTAPPROXIMATIONS Plots the different approximations of interest

movie\_size = size(A);

len = movie\_size(2);

t = (1:len)./20;

subplot(6,3,1)

A = (A+mean\_A).\*(len-1);

plot3(A(1,:),A(2,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('Original Cam 1')

subplot(6,3,2)

plot3(A(3,:),A(4,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('Original Cam 2')

subplot(6,3,3)

plot3(A(5,:),A(6,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('Original Cam 3')

approx1 = (u(:,1)\*s(1,1)\*v(:,1)' + mean\_A).\*(len-1);

subplot(6,3,4)

plot3(approx1(1,:),approx1(2,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('1st Order Approx Cam 1')

subplot(6,3,5)

plot3(approx1(3,:),approx1(4,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('1st Order Approx Cam 2')

subplot(6,3,6)

plot3(approx1(5,:),approx1(6,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('1st Order Approx Cam 3')

mode2 = (u(:,2)\*s(2,2)\*v(:,2)' + mean\_A).\*(len-1);

subplot(6,3,7)

plot3(mode2(1,:),mode2(2,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('2nd Mode Cam 1')

subplot(6,3,8)

plot3(mode2(3,:),mode2(4,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('2nd Mode Cam 2')

subplot(6,3,9)

plot3(mode2(5,:),mode2(6,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('2nd Mode Cam 3')

approx2 = (u(:,1:2)\*s(1:2,1:2)\*v(:,1:2)' + mean\_A).\*(len-1);

subplot(6,3,10)

plot3(approx2(1,:),approx2(2,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('2nd Order Approx Cam 1')

subplot(6,3,11)

plot3(approx2(3,:),approx2(4,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('2nd Order Approx Cam 2')

subplot(6,3,12)

plot3(approx2(5,:),approx2(6,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('2nd Order Approx Cam 3')

mode2 = (u(:,3)\*s(3,3)\*v(:,3)' + mean\_A).\*(len-1);

subplot(6,3,13)

plot3(mode2(1,:),mode2(2,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('3rd Mode Cam 1')

subplot(6,3,14)

plot3(mode2(3,:),mode2(4,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('3rd Mode Cam 2')

subplot(6,3,15)

plot3(mode2(5,:),mode2(6,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('3rd Mode Cam 3')

approx2 = (u(:,1:3)\*s(1:3,1:3)\*v(:,1:3)' + mean\_A).\*(len-1);

subplot(6,3,16)

plot3(approx2(1,:),approx2(2,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('3rd Order Approx Cam 1')

subplot(6,3,17)

plot3(approx2(3,:),approx2(4,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('3rd Order Approx Cam 2')

subplot(6,3,18)

plot3(approx2(5,:),approx2(6,:),t)

set(gca,'Ylim',[100 400],'Xlim',[200 500])

title('3rd Order Approx Cam 3')

end

Homework3.m

%% Start with a clean slate

clear all; close all; clc

%% Video 1

load('cam1\_1.mat');

load('cam2\_1.mat');

load('cam3\_1.mat');

% First, try to get a grasp of what we're looking at

%movie\_size = size(vidFrames1\_1);

% We're dealing with a 4-d data set that is 480 640 3 226 in size

% We know it's a video, so most like 480x640 pixels, 226 measurements, and

% the 3 is for the RGB values. What does a random frame look like?

%image(vidFrames1\_1(:,:,:,200))

% What does the overall video look like?

%implay(vidFrames1\_1)

% the cams have different number of frames, manually cut the data to start

% at the same time - I used the frame before I see first downward movement

[vidFrames1\_1,vidFrames2\_1,vidFrames3\_1] = AlignVideos(vidFrames1\_1,11,vidFrames2\_1,20,vidFrames3\_1,11);

% the flashlight isn't the only thing that's white so I had more

% success tracking the bright yellow of the pain can

[cam1\_x,cam1\_y,cam1\_corrupt] = TraceColor(vidFrames1\_1, 250, 256, 240, 250, 200, 210);

[cam2\_x,cam2\_y,cam2\_corrupt] = TraceColor(vidFrames2\_1, 160, 200, 175, 220, 60, 120);

% cam 3 was very difficult due to the turning of the can and blur in

% frames 227 and 228. ExtractCommonColors helped find a good color that

% appears across all frames, it takes a while to run though

% commonColorRanges = ExtractCommonColors(vidFrames3\_1);

[cam3\_x,cam3\_y,cam3\_corrupt] = TraceColor(vidFrames3\_1, 185, 200, 185, 205, 130, 150);

% sanity check visually the tracking

% implay(cam1\_corrupt)

A = [cam1\_x;cam1\_y;cam2\_x;cam2\_y;cam3\_x;cam3\_y];

movie\_size = size(A);

% normalize data

A = A./(movie\_size(2)-1);

mean\_A = mean(A,2);

A = A - mean\_A;

[u,s,v] = svd(A);

figure(1)

PlotApproximations(A,u,s,v,mean\_A);

figure(2)

sig=diag(s);

subplot(1,2,1), plot(sig,'ko','Linewidth',[1.5])

subplot(1,2,2), semilogy(sig,'ko','Linewidth',[1.5])

%% Video 2

load('cam1\_2.mat');

load('cam2\_2.mat');

load('cam3\_2.mat');

[vidFrames1\_2,vidFrames2\_2,vidFrames3\_2] = AlignVideos(vidFrames1\_2,13,vidFrames2\_2,38,vidFrames3\_2,18);

[cam1\_x,cam1\_y,cam1\_corrupt] = TraceColor(vidFrames1\_2, 215, 240, 190, 210, 140, 160);

[cam2\_x,cam2\_y,cam2\_corrupt] = TraceColor(vidFrames2\_2, 200, 240, 230, 256, 160, 190);

[cam3\_x,cam3\_y,cam3\_corrupt] = TraceColor(vidFrames3\_2, 230, 256, 230, 256, 180, 210);

A = [cam1\_x;cam1\_y;cam2\_x;cam2\_y;cam3\_x;cam3\_y];

movie\_size = size(A);

% normalize data

A = A./(movie\_size(2)-1);

mean\_A = mean(A,2);

A = A - mean\_A;

[u,s,v] = svd(A);

figure(3)

PlotApproximations(A,u,s,v,mean\_A);

figure(4)

sig=diag(s);

subplot(1,2,1), plot(sig,'ko','Linewidth',[1.5])

subplot(1,2,2), semilogy(sig,'ko','Linewidth',[1.5])

%% Video 3

load('cam1\_3.mat');

load('cam2\_3.mat');

load('cam3\_3.mat');

[vidFrames1\_3,vidFrames2\_3,vidFrames3\_3] = AlignVideos(vidFrames1\_3,20,vidFrames2\_3,8,vidFrames3\_3,12);

[cam1\_x,cam1\_y,cam1\_corrupt] = TraceColor(vidFrames1\_3, 215, 235, 140, 160, 135, 160);

[cam2\_x,cam2\_y,cam2\_corrupt] = TraceColor(vidFrames2\_3, 200, 240, 230, 256, 160, 190);

[cam3\_x,cam3\_y,cam3\_corrupt] = TraceColor(vidFrames3\_3, 230, 256, 231, 256, 180, 210);

A = [cam1\_x;cam1\_y;cam2\_x;cam2\_y;cam3\_x;cam3\_y];

movie\_size = size(A);

% normalize data

A = A./(movie\_size(2)-1);

mean\_A = mean(A,2);

A = A - mean\_A;

[u,s,v] = svd(A);

figure(5)

PlotApproximations(A,u,s,v,mean\_A);

figure(6)

sig=diag(s);

subplot(1,2,1), plot(sig,'ko','Linewidth',[1.5])

subplot(1,2,2), semilogy(sig,'ko','Linewidth',[1.5])

%% Video 4

load('cam1\_4.mat');

load('cam2\_4.mat');

load('cam3\_4.mat');

[vidFrames1\_4,vidFrames2\_4,vidFrames3\_4] = AlignVideos(vidFrames1\_4,15,vidFrames2\_4,21,vidFrames3\_4,14);

% this video is long enough with enough sides of the can seen that no

% matter what color I picked, there would be some frames without the color,

% so using color that tracks the most and fixing the untagged frames

[cam2\_x,cam2\_y,cam2\_corrupt] = TraceColor(vidFrames2\_4, 200, 240, 220, 256, 150, 185);

cam2\_x(14) = 364; cam2\_y(14) = 275;

[cam3\_x,cam3\_y,cam3\_corrupt] = TraceColor(vidFrames3\_4, 190, 220, 190, 220, 140, 160);

cam3\_x(168) = cam3\_x(167); cam3\_y(168) = cam3\_y(167);

cam3\_x(242) = 330; cam3\_y(242) = 207;

cam3\_x(243) = 330; cam3\_y(243) = 207;

% cam 1 was especially difficult, so I'm blanking out areas that never have

% the can, and then fixing up several frames

corrupt\_cam1 = vidFrames1\_4;

corrupt\_cam1(:,470:640,:,:) = 0;

corrupt\_cam1(:,1:320,:,:) = 0;

corrupt\_cam1(400:480,:,:,:) = 0;

corrupt\_cam1(1:230,:,:,:) = 0;

[cam1\_x,cam1\_y,cam1\_corrupt] = TraceColor(corrupt\_cam1, 190, 230, 190, 220, 140, 180);

cam1\_x(10) = 396; cam1\_y(10) = 396;

cam1\_x(11) = 396; cam1\_y(11) = 396;

A = [cam1\_x;cam1\_y;cam2\_x;cam2\_y;cam3\_x;cam3\_y];

movie\_size = size(A);

% normalize data

A = A./(movie\_size(2)-1);

mean\_A = mean(A,2);

A = A - mean\_A;

[u,s,v] = svd(A);

figure(7)

PlotApproximations(A,u,s,v,mean\_A);

figure(8)

sig=diag(s);

subplot(1,2,1), plot(sig,'ko','Linewidth',[1.5])

subplot(1,2,2), semilogy(sig,'ko','Linewidth',[1.5])