AMATH 482 Winter 2020

Homework 3:Principal Component Analysis

Tanner Graves

Introduction and Overview

Abstract

This assignment presented the task of firstly using image and video analysis to obtain position data in a series of videos; a paint can has been filmed from a variety of different perspectives under several scenarios oscillating on an elastic cord. we ‘re given multiple different camera angles - including not upringt, one oscillating vertically under ideal conditions, one with added noise in the form of violent camera movement, another with significant horizontal movement as well as vertical, and lastly one that has been perturbed horizontally, vertically and rotationaly. Once this data has been obtained we may begin to analyze it for redundancies using Principal Component Analysis to yield more useful physical information about the system.

Overview

Though I could ask a person where they see a can of paint at any particular moment, asking a computer to do so proves less trivial. If it is a certain object we’re looking for, we can start asking some questions like “What color is it?”, but trying to recognize the same object across multiple points in time and register its movement, is rather unintuitive. Luckily there's many tools available to aid in this. Specifically, being able to utilize threshold filters and a point tracking algorithm to extract position data.

Once position is obtained, it will be fairly redundant, as we will have 3 sets of 2 dimensional data to describe something that is happening fundamentally in 3. We will be able to use some methods from statistics, specifically variance as a measure of this redundancy. Additionally, we’ll be able to employ some heavier tools from linear algebra to aid in interpreting the data and further identifying redundancies, namely, singular value decomposition, or SVD.

SVD, simply put, will yield an alternative representation of our data set, where we will be able to get information of the stretch and angle trends that exist inside the data. A nice result of this method is it gives us a handle on the “energy “ in a system. For this physical application, this will give us an insight into the dynamics of our system.

Theoretical Background

Image processing

A large part of dealing with systems represented by large data sets is identifying what actually influences our system. Having concrete answers to the question “what data is actually useful?” can greatly aid in computation if it allows the scope of considered data to be limited. We will greatly be aided by tools from statistics in doing this. We may quantitatively measure the variance,or loosely how much a data set is spread out, and the covariance, or the correlation of two different data sets and are given by the equations 1 and 2 respectively:

$$\sigma^2 = \frac{1}{n-1} aa^T$$

$$\sigma^2 = \frac{1}{n-1} ab^T$$

Given many different data sets we may use the covariance matrix, to measure the correlation across many different parameters

$$C_X = \frac{1}{n-1} XX^T$$

Interpreting this square matrix is easy once noting that variances are along the diagonal and each off diagonal entry is a covariance between the parameters that correspond to the row and column of that entry.

From a linear algebra prospective, Singular Value Decomposition, or SVD can be seen to take a mxn matrix and represent it as the product 3 different matrices, each with information that is desirable for interpretation of data. These matrices are U, a unitary mxn matrix; Σ a diagonal matrix with positive entries; and V, a unitary nxn matrix. An important property of SVD is the columns of U are orthonormal, the diagonal entries of Σ are scalars that act on the columns of U to form the principal semiaxes.

Returning to our original question, we can now make use of SVD for the purposes of dimensionality reduction. Since we may interpret the σs as the amount of energy captured by a principal axis, we may use this information to construct an optimal approximation of our matrix(data) of a lesser order. By optimal I mean that the Frobenius norm, is minimized for a given approximation of any order. Further information can be gotten through the process of diagonalization, or expressing our original data set in terms the basis provided to us by either U or V.

Algorithm Implementation and Development

A large portion of code development was spent trying to find elegant solutions to the problem of processing the video. I quickly realized that tracking the flashlight would be the most convenient approach, as the intensity of the light is a very detectable feature to track - a simple threshold for whites, above a certain intensity gave satisfactory isolation.

The issue with this approach comes when the front of the lite rotates out of view and we are instead faced with only the pink back side of the light. Detecting the pink then would then be able to give us information about the location of the same object during these times.

My first approach was to attempt to make a similar rudimentary series of thresholds to isolate the pink. I could sample frames in which the back was visible and take the values of pixels that should be included in the threshold inequalities. Even with taking many samples, this approach failed to yield satisfactory results. Often too much background was included, or too much of the light was excluded. Feeling that my approach was too simple, I aimed to find a more sophisticated filtering technique. I began by taking a series of sample frames from each video, and then creating a mask in MSPaint of all the pixels I considered desirable - either the white of the light or the pink of the body. After collecting a fair amount of data, I plotted these points in color space, where each axis represents the intensity of either the red, green, or blue component of each pixel that was positively identified in the masking process. The result was a cloud of points. Seeing this, it is easy to tell how the original threshold approach was too simple.

To use this data to implement a filter, we are now faced with the problem of giving an arbitrary RGB color, determining if it lies within our point cloud. To answer this question in the most direct way possible I made a surface of the boundary of the point cloud. I may then use the workshop function inPolyhedron (citation). To determine if or color vector is within this hull. This turns out to be prohibitively slow for the application of image analysis due to inpolyhedron having bad behavior the more surfaces (correlated to the number of sampled points on the boundary of the point cloud) you have. Using something like reducepatch to aid in this still proves taxing and introduces error. My final approach along this tangent was to approximate the the boundary of the point cloud with a polynomial in three variables: r, g, and b. Similar to how in R^2 we can express all points inside a circle by 1>x^2 + y^2, we may express the volume bounded by a surface by an inequality involving a polynomial of 3 variables. Approximating the boundary points with a polynomial is actually a neat application of SVD, where I can

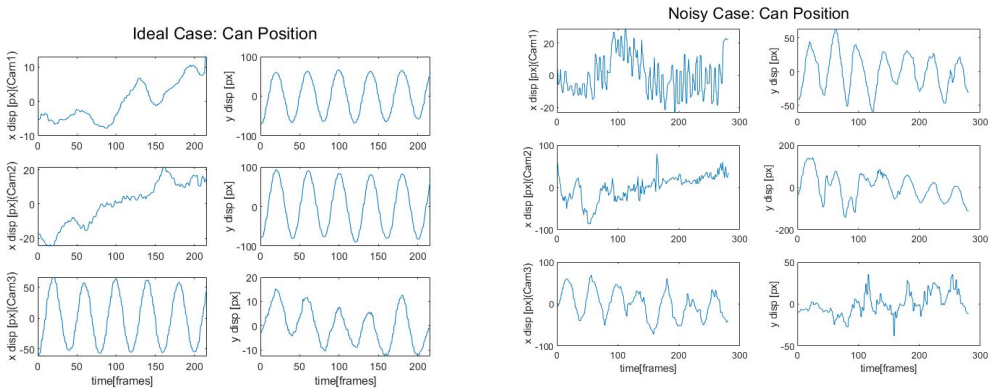
use the analysis to determine the best n-order polynomial approximation of the set of points. Additionally, since polynomials are really nice to compute, this approach has much promise for filter application.

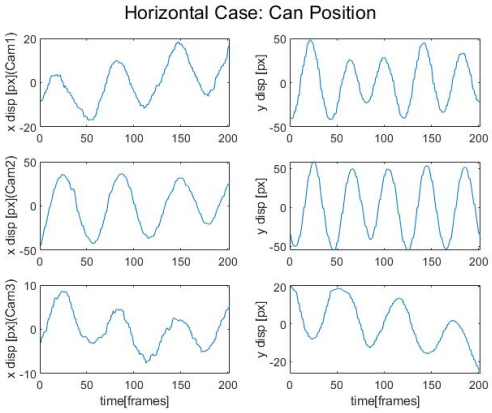
Even though the RGB color space is a perfectly fine representation of color, it turns out not to gybe with your intuition of color very well. It proved much simpler to use the Image Processing toolbox to convert the image into Hugh, Saturation, Value (HSV) data and filter that way. This proved a far more effective filtering tool and the tools available made it quite easy to get hands on with data.

Computational Results

Image Processing

By applying a variety of threshold masks and tracking various points on the paint can throughout its motion I was able to derive its motion for the given cases. The graph shows the X and Y position of the can as seen by each camera.





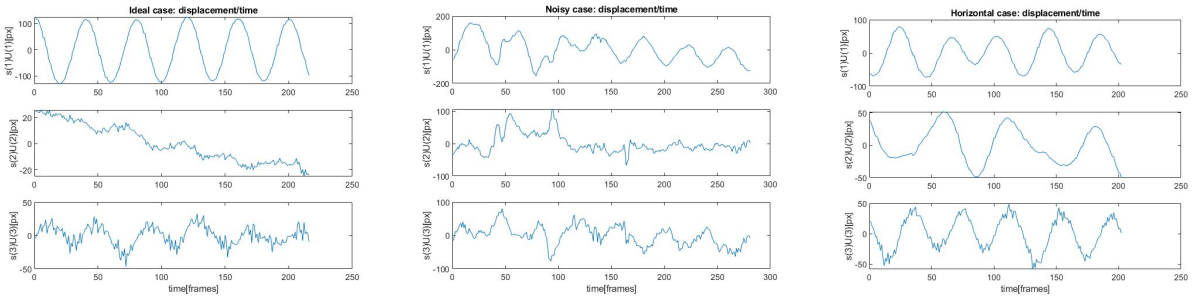
Now that we have position data, we can start using tools from Principal component analysis. As an example: the covariance matrix for the ideal case is as follows:

1.0e+03 *

0.0321	0.0011	0.0530	-0.0307	-0.0082	-0.0110
0.0011	2.0852	-0.0944	2.6670	1.8618	0.2827
0.0530	-0.0944	0.1633	-0.1973	-0.1143	-0.0532
-0.0307	2.6670	-0.1973	3.5317	2.4017	0.3838
-0.0082	1.8618	-0.1143	2.4017	1.7101	0.2663
-0.0110	0.2827	-0.0532	0.3838	0.2663	0.0553

Observing the off-diagonal entries, there is a fair amount of redundancy present in this system. We can see that row 2 column 4 has a large entry. We may compare this to the second and fourth graphs in the figure: Ideal case: Can Position, to see they indeed have a strong correlation.

After taking the SVD, we can get an idea of what's going in the orthonormal basis that SVD provides. Below are graphs of the highest three σs multiplied by their respective U column.



Using SVD I was able to arrive at the the following singular values:
10^3*

Ideal	Noisy	Horizontal
1.2520	1.1820	628.7511
0.2065	0.4970	374.0040
0.1054	0.4010	162.7956
0.0631	0.3025	101.9520
0.0540	0.1934	45.6882
0.0312	0.1759	29.0646

This gives a measure to the energy contained in the motion on each axis in the basis. And, as can be observed in the dropoff of values, there is indeed redundancy in the system

Summary and conclusion

Through the use of computer vision and image processing we have managed to determine the position of a moving object despite considerable variations in the data(noise, rotation, etc). This data proved redundant, something which we were able to quantify through the use of PCA. Additionally, we can get an idea of some of the dynamics of the system through the singular values.

PCA proved to be a powerful tool for dimensionality reduction and data interpretation.

Appendix A. MATLAB Functions

```
svd(A,'econ') - reduced Singular value decomposition
App: Image thresholder - useful for utilizing different color spaces
vision.pointTracker - a powerful and computationally effective way of
getting movement data out of a series of images
Inpolyhedron - given a closed surface can determine if a given point
is bounded
Boundary - given a set of points, finds the subset that is on the
boundary of the group of points
```

Appendix B. MATLAB Code and Data

horizCase.m (same code for the other cases, but different filter values)

```
%% Load and play video
clear all;close all;clc;
load('cam1_3.mat');load('cam2_3.mat');load('cam3_3.mat')
I13 = vidFrames1_3;
I23 = vidFrames2_3;
I33 = vidFrames3_3;
n13 = size(I13,4);
n23 = size(I23,4);
n33 = size(I33,4);
I13f = msk13p(I13(:,:,:,1));
I23f = mask23w(I23(:,:,:,1));
I33f = I33(:,:,:,1);
%%
tk13 = vision.PointTracker;
initialize(tk13,[328 238],255*I13f);
pts13 = zeros(n13,2);
for j = 1:n13
    X = msk13p(I13(:,:,:,j));
    [pts13(j,:),~] = tk13(255*X);
end

tk23 = vision.PointTracker;
initialize(tk23,[250 371],255*I23f);
pts23 = zeros(n23,2);
for j = 1:n23
    X = mask23w(I23(:,:,:,j));
    [pts23(j,:),~] = tk23(255*X);
    %imshow(255*X);
    %pause(0.2);
    pts23(j,:);
end

tk33 = vision.PointTracker;
initialize(tk33,[248 230],255*I33f);
pts33 = zeros(n33,2);
for j = 1:n33
    %X =
msk31w(I32(:,:,:,j))|msk31s(I32(:,:,:,j))|msk31m(I32(:,:,:,j));
    %X = msk321(I32(:,:,:,j));
    %X = msk33w(I33(:,:,:,j));
    X = I33(:,:,:,j);
    [pts33(j,:),~] = tk33(X);
```

```
end
%% filter troubleshooting
figure(1)
for j = 1:n33
    %X =
msk31w(I32(:,:,:,j))|msk31s(I32(:,:,:,j))|msk31m(I32(:,:,:,j));
    X = msk33w(I33(:,:,:,j));
    imshow(255*X);
    pause(0.01);
end

%% line up the data in time
A1 = pts13(38:end,:);
A2 = pts23(61:61+201,:);
A3 = pts33(28:28+201,:);
A = [A1 A2 A3];
Am = mean(A,1);
A = A-Am;
[U,S,V] = svd(A,'econ');
s = diag(S);
A3 = zeros(size(A));
for j = 1:3
    A3 = A3 + s(j)*U(:,j)*V(:,j)';
end

%% plot: Horizontal Case
figure(1)
sgtitle('Horizontal Case: Can Position');
subplot(3,2,1)
plot(A(:,1))
ylabel('x disp [px](Cam1)')
subplot(3,2,2)
plot(A(:,2))
ylabel('y disp [px]')
subplot(3,2,3)
plot(A(:,3))
ylabel('x disp [px](Cam2)')
subplot(3,2,4)
plot(A(:,4))
ylabel('y disp [px]')
subplot(3,2,5)
plot(A(:,5))
ylabel('x disp [px](Cam3)')
```

```
xlabel('time[frames]')
subplot(3,2,6)
plot(A(:,6))
ylabel('y disp [px]')
xlabel('time[frames]')
%% plot: Horiz Case
figure(1)
subplot(3,1,1);
plot(s(1)*U(:,1))
title('Horizontal case: displacement/time')
ylabel('s(1)U(1)[px]')
subplot(3,1,2);
plot(s(2)*U(:,2))
ylabel('s(2)U(2)[px]')
subplot(3,1,3);
plot(s(2)*U(:,3))
ylabel('s(3)U(3)[px]')
xlabel('time[frames]')
```