EEL 6562
Course Project

Intelligent Hand Recognition and Tracking

Group Members:
Daniel Schoonover

Abstract

Human-computer interfaces have seen little advancement over the past decade. Even today, computer users are still restricted to the keyboard/mouse human input method. There is much room for intuitive improvement in this area, and our project will address this task. I proposed to utilize the abundance of inexpensive web cams to implement a computer vision algorithm that recognizes and tracks hand movements. By recognizing the location of human hands in the image feed, the algorithm will first make a decision as to whether there is a hand on-screen and the relative location of that hand. Next, the hand will be tracked as it moves along the screen. The algorithm itself is the ultimate goal of the project, although the obvious application for this is the real-time updating of screen pointer data to be able to move the mouse with hand gestures. Some stretch goals outside the scope of this project are to extend the object recognition algorithm to understand different hand gestures to allow for clicking/ right clicking and the porting of the program to a less computationally-demanding language than MATLAB.

Introduction

The problem at hand was to find an efficient technique for object recognition. This could have been done in a number of different ways, but the most advanced hierarchal method I chose to use was Principle Component Analysis, also known as the Karhunen-Loève Transform. Principle Component Analysis (PCA) is used to determine the principle components of a training data set by finding the eigenvectors of the covariance matrix of the training set. PCA is explained in full glorious detail in the 'Description' section below.

As an engineer, the tradeoff I had to optimize was the utilization of the least number of eigenvectors (to decrease computation time) while still maintaining enough information to successfully track the hand. Additionally, I had to find a way to reduce the raw image data by low-level methods to quickly and cheaply find the regions of the image that could contain the hand. A combination of techniques were used here, the main technique being hue filtering to detect skin regions.

Description

Given a raw video input, the purpose of my project was to detect the user's hand in or close to real time. This problem was divided into 2 nearly distinct parts. The first part was segmentation of the raw image based on lower-level analysis into smaller image segments, each of which could contain the hand. The second part was higher level analysis (principle component analysis to be exact) of those image segments to decide which segment did contain the hand.

Part 1) Raw Image Segmentation

In the image segmentation step, the goal was to take a raw image feed, and create a filter to dissect it into smaller regions, each of which had near-equal probability of containing the hand. For this step, RGB images were converted into HSV hue, which made skin tone analysis easy, although it should be noted that the filter had to be calibrated based on the user's skin tone.

The filter used a mask to remove all non-skin elements from the image, and the BWAREAOPEN function was used to eliminate regions smaller than what was considered to be suitably hand-size. Also, the mask was closed using morphological methods to remove speckling caused by inconsistencies in lighting(and skin blemishes). The largest regions that passed both the skin hue and region size specifications were found to contain one of two possible regions: the users face or the users hand.

MATLAB code for image segmentation

```
%H1 is the raw image from the video feed

    %Filter skin color:
    h = rgb2hue(single(H1));
    mask = ones(size(h));
    mask((h>47.5 & h<260)) = 0; %depend on lighting & skin color

    %despeckle by closing the mask
    se = strel('disk', 2, 0);
    mask = imerode(mask, se);
    mask = imdilate(mask, se);

    mask = bwareaopen(mask,2700);
    mask = imfill(mask, 'holes');

    %prepare image for presentation
    L = bwlabel(mask);
    img = gray.* uint8(mask);
    img = 3.25 * (img - 80);
    figure(1), imshow(img);
```

```
% Draw bounding boxes and centroids
bb = regionprops(L, 'BoundingBox');
c = regionprops(L, 'Centroid');
hold on

for k = 1:numel(bb)
    x(1) = bb(k).BoundingBox(1);
    y(1) = bb(k).BoundingBox(2);
    x(2) = x(1) + bb(k).BoundingBox(3);
    y(2) = bb(k).BoundingBox(2);
    x(3) = x(1) + bb(k).BoundingBox(3);
    y(3) = y(1) + bb(k).BoundingBox(4);
    x(4) = bb(k).BoundingBox(1);
    y(4) = y(1) + bb(k).BoundingBox(4);
    patch(x, y, 'g', 'FaceAlpha', 0.15)

    x(1) = c(k).Centroid(1) - 3;
    x(2) = c(k).Centroid(1) + 3;
    x(3) = c(k).Centroid(1) + 3;
    x(4) = c(k).Centroid(1) - 3;
    y(1) = c(k).Centroid(2) - 3;
    y(2) = c(k).Centroid(2) - 3;
    y(3) = c(k).Centroid(2) + 3;
    y(4) = c(k).Centroid(2) + 3;
    patch(x, y, 'r')
end
```

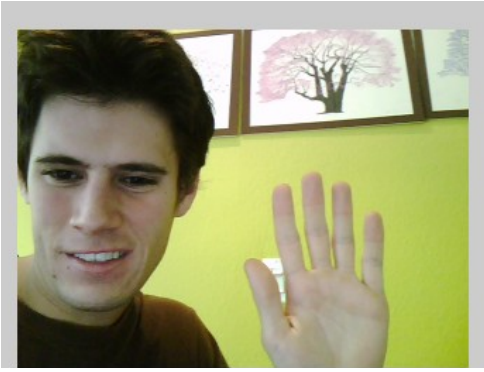

Illustration 1: Raw test image

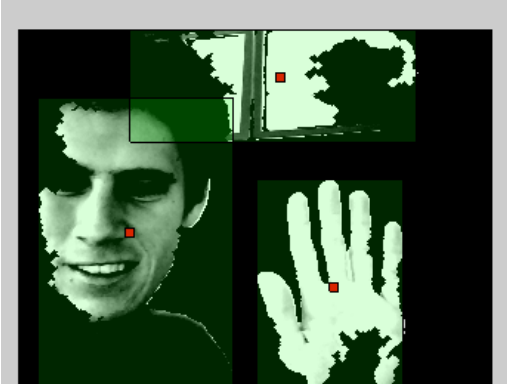

Illustration 2: Skin filtering of raw image

As can be seen, the skin filter segments each image into regions that have 'high hand probability'. From here, the image segments are passed into the next stage of detection, principle component analysis.

Part 2) Principle Component Analysis

In PCA, a data set or library is created from images that represent closely the image features that are important. In my case, I constructed a library of hand images. All of these images were then placed in a (N,S) matrix where N is the number of images in the library and S is the image size (length * width). Next I found the eigenvectors and corresponding eigenvalues of the covariance matrix of this data set.

The highest eigenvalues represented the eigenvectors that corresponded to the maximum variance between each image in the training set. I chose to use the 10 'best' eigenvectors, and their projection into image space I called the 'eigenhand'.

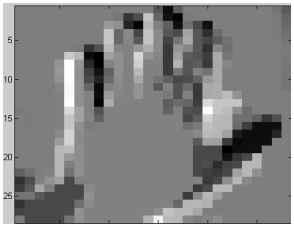

Illustration 3: The Eigenhand

To decide which image segment actually contained the hand, the eigenvector projection coefficient vector was found for each image and their absolute distances were compared. The coefficient vector C was calculated as

$$C = T * p^T$$

where p is the image being tested in vector form, and T is the PCA transformation matrix, composed of the eigenvectors of the covariance matrix of the hand library. The size of the coefficient vector is Nx1, where N is the number of eigenvectors used from the data set. From here, each image within the library is projected into this feature space, and a 'mean coefficient vector' is calculated. To decide which image segment contains the most 'hand-ness', each segment is projected into the feature space, and the resulting vectors are compared to the mean coefficient vector. The vector that is closest to the mean coefficient vector is decided to be the hand.

The PCA algorithm was tested on the raw image shown above for each of the image segments. The distances of each image segment from the mean coefficient vector are shown below:

distance of hand projection from mean coefficient vector
ans = 5.0586

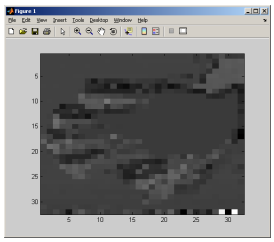

Illustration 4: Projection of hand image segment onto feature space

distance of face projection from mean coefficient vector
ans = 49.7303

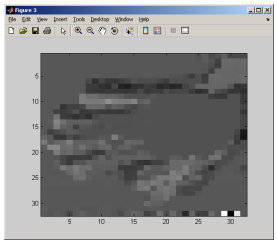

Illustration 5: Projection of face image segment onto feature space

distance of blob projection from mean coefficient vector
ans = 332.3470

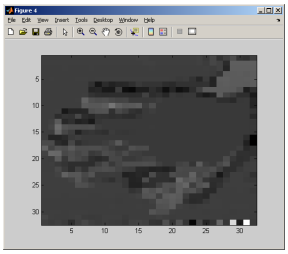

Illustration 6: Projection of unknown image onto feature space

As can be seen, the image segment containing the hand had the least variation from the mean coefficient vector, and the unknown 'blob' image had the most variation. With these projections, it was simple to make the decision as to which image segment contained the hand.

MATLAB code for PCA:

```
h1,h2,h3 and h4 are the hand images used to create the data set.
Face, handt and falseim are the 3 image segments created with the skin filtering
analysis, (above)

%Calculate mean coefficient vector by projecting each of the training images
%onto the 10 best eigenvectors of the training set:
Prej1 = EVecs(:,1:10)'*double(h1(:)-mean(mean(h1))); % Project onto ev's 1-10
Prej2 = EVecs(:,1:10)'*double(h2(:)-mean(mean(h2))); % Project onto ev's 1-10
Prej3 = EVecs(:,1:10)'*double(h3(:)-mean(mean(h3))); % Project onto ev's 1-10
Prej4 = EVecs(:,1:10)'*double(h4(:)-mean(mean(h4))); % Project onto ev's 1-10
%now get mean of the projection vectors
MeanCoVec=(Prej1+Prej2+Prej3+Prej4)/4;


%calculate projection of test images & the distance of the resulting vectors from
% the mean coefficient vector, and display each projection:
Prejhand = EVecs(:,1:10)'*double(handt(:)-mean(mean(handt))); % Project hand
test image
%project coeff vector onto feature space:
showvec1=EVecs(:,1:10)*Prejhand;
figure();imagesc(reshape(showvec1,h,w)');colormap(gray(256));
'distance of hand proj from mean coefficient vector'
abs(sum(Prejhand-MeanCoVec))


Prejface = EVecs(:,1:10)'*double(face(:)-mean(mean(face))); % Project face test
image
%project coeff vector onto feature space:
showvec2=EVecs(:,1:10)*Prejface;
figure();imagesc(reshape(showvec2,h,w)');colormap(gray(256));
'distance of face proj from mean coefficient vector'
abs(sum(Prejface-MeanCoVec))

Prejfalseim = EVecs(:,1:10)'*double(falseim(:)-mean(mean(falseim))); % Project
face test image
%project coeff vector onto feature space:
showvec3=EVecs(:,1:10)*Prejfalseim;
figure();imagesc(reshape(showvec3,h,w)');colormap(gray(256));
'distance of blob proj from mean coefficient vector'
abs(sum(Prejfalseim-MeanCoVec))
```

Evaluation

The methods used in our scripts worked well, and in near-real time. Considering the performance of MATLAB to other programming languages, the results of this paper are sufficient to assume that the algorithm works quickly and sufficiently enough to be used by most modern computers. The skin filtering algorithm used in this paper worked in real time(roughly 250ms per each image filtered, including image acquisition and downsampling), and did a good job in segmenting the regions in an image that contained skin-like regions. The downside to the method we used was that thresholds had to be calibrated for users with different skin color, and for different lighting schemes. This problem could have been avoided by creating a user-calibration script to be run before the algorithm, given more time and manpower on working on this project.

PCA proved to be an easy way to detect features although it was somewhat slow for large image sizes. For each image being tested for 'hand-ness', the algorithm had to project the image onto the same number of eigenvectors as the image had pixels. This downside was easily circumvented by downsizing the image in question (and the hand library) to greatly reduce computation time. Assuming the data set is computed offline, this segment of code took only 50 ms to run in MATLAB, for 3 possible image segments, and downsizing the image to a 32*32 pixel size.

Summary and Conclusions

This project was one of the most fun MATLAB projects I have ever worked on, and considering my enthusiasm for programming, this says a lot. I learned how to acquire a real time video feed into MATLAB from a laptop webcam, and how to filter these images in real time, segmenting images into skin-like segments. Moreso, I finally learned how Principle Component Analysis projects images into a feature space, which is a very powerful data analysis tool. I think more than anything, I learned how important it is to have friends with great coding skills.

Related work/Reference List

There have been a number of papers utilizing principle component analysis for image recognition, although this is the first using it to perform hand recognition. The majority of papers using PCA for recognition did so for facial recognition, that is, to match a new face to a training set of face images. These papers helped narrow down the research process. Some of these papers (and/or webpage tutorials) are linked below in ascending order of help provided to this project:

[1] “Matt's Matlab Tutorial Source Code Page” http://www.cs.ait.ac.th/~mdailey/matlab/

[2]Jon Krueger, Marshall Robinson, Doug Kochelek, Matthew Escarra. “Obtaining the Eigenface Basis”. http://cnx.org/content/m12531/latest/

[3] Dimitri Pissarenko “Eigenface-Based Facial Recognition” http://openbio.sourceforge.net/resources/eigenfaces/eigenfaces-html/facesOptions.html