Discrete Gaze-Estimation using Machine Learning

Ahmad Salim Al-Sibahi and Nicolai Skovvart*

IT University of Copenhagen, Denmark

* Supervisor: Dan Witzner Hansen

Abstract. Abstract goes here

Keywords: Batman

Table of Contents

1 Introduction

We will perform an experiment where a subject is looking at a finite amount of points under different conditions. These conditions are combinations of moving the head or keeping it still and an infrared light being on or off. The images will then be processed to identify the edges of the eye and the center of the pupil. We can then extract eye images using the feature data. We can then run machine learning algorithms on the data and analyze the results to determine if we are able to perform gaze estimation.

This report will explain concepts of machine learning utilized in this project, the experiment in more detail, how the results can be evaluated, possible threats to validity and the final results.

2 Problem Description

In this project we will determine under what conditions it is possible to perform gaze estimation using machine learning.

Gaze estimation is the process of identifying where an eye is looking. It has a lot of applications, for example in giving disabled people the ability to type with, and control devices using, their eyes. Especially people with Amyotrophic Lateral Sclerosis (ALS) can make use of this technology. It is also closely related to eye tracking where image analysis is used to identify eyes and eye features such as eye corners, center of the iris, glints in the eye and so on. We will not be using eye tracking as it is outside the scope of this project.

Using machine learning for gaze estimation can be problematic, as images of faces (or eyes) are high-dimensional data, making learning complex and expensive. There are a few approaches to reducing the complexity. One could reduce the input space by converting the image to grayscale, isolating the eye-pixels from the rest of the image, and scaling the resulting image. One could also reduce the input space by extracting features of the eye such as the positions of the corners of the eye and the center of the pupil. In this project we attempt both approaches.

When gathering test data there are also many factors to consider. Test subjects should be placed similarly in such a way that their eyes are clearly visible to the camera and at a distance where the glints in the eyes from the infrared lights are visible. The angle of the head and eyes also impacts the resulting images. Lighting is also very important and should be consistent during testing.

When processing the initial test data, it can be beneficial to histogram equalize the images to increase the contrast. This makes lighting differences have less of an impact.

3 Theory

Theory goes here.

3.1 Principal Component Analysis

Principal Component Analysis (PCA) is a statistical technique that can be used to identify patterns in high dimensional data, for example images. Examples in this section are inspired by the PCA tutorial by Smith [1].

3.1.1 How To Use

Input data formatting. First of all, the data should be prepared for PCA. In the case of images, the image should be flattened to a single vector. This means an image of size M by N with Z dimensions (colour values) should turn into a vector with length $M \times N \times Z$. Turning the images into grayscale are often a good idea as they reduce the complexity by a third, and the added colour information may not be valuable. A mean vector of all the input vectors should then be calculated and subtracted from the input vectors, producing a new input set with a mean of 0. For 2 grayscale images of size 2×2 pixels, the process could look like table 1. All of the adjusted input vectors should then be placed in a matrix, where every row is an adjusted image vector.

Table 1. Input data formatting example

Vector	(X_1, Y_1)	(X_2, Y_1)	(X_1, Y_2)	(X_2, Y_2)
$Image_1$	150	220	123	136
$Image_2$	20	110	240	11
Mean	85	165	181.5	73.5
$AdjustedImg_1$	65	55	-58.5	62.5
$AdjustedImg_2$	-65	-55	58.5	-62.5
AdjustedMean	0	0	0	0

Calculate the covariance matrix. The covariance between two dimensions can be defined as follows, assuming the mean has already been subtracted from X and Y.

$$cov(X, Y) = \frac{\sum_{i=1}^{n} X_i Y_i}{n - 1}$$

The covariance should be calculated for all dimensions. For example, for a 3 dimensional data set with dimensions (x, y, z) the covariance can be calculated for (x, y), (x, z) and (y, z). The covariance matrix for a data set with n dimensions is defined as follows.

$$C^{n \times n} = (c_{i,j}, c_{i,j} = cov(Dim_i, Dim_j))$$

The covariance matrix for the previous imaginary data set with dimensions (x, y, z) is the following.

$$C = \begin{pmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{pmatrix}$$

The matrix is a square $n \times n$ matrix, and is symmetrical around the main diagonal, as $cov(a, b) = cov(b, a)$. It is also worth noting that down the main diagonal the value is the covariance between a dimension and itself.

Calculate eigenvectors and eigenvalues of the covariance matrix. Eigenvectors are vectors that when multiplied with another matrix work like a constant. For example:

$$\begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \times \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 12 \\ 8 \end{pmatrix} = 4 \times \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

Some properties of eigenvectors: eigenvectors of a matrix can only be found for square matrices, and not every square matrix has eigenvectors. Given an $n \times n$ matrix that does have eigenvectors, there are n of them. There is no easy way to calculate eigenvectors, but most programming languages have libraries with support for calculating them.

Eigenvalues are closely related to eigenvector and there was one in the previous example, namely 4. Eigenvalues comes in pairs with eigenvectors.

Choosing components. The eigenvector with the highest eigenvalue is the principle component of the data set. Sorting the eigenvectors by eigenvalue allows us to see what components are most important, and allows us to ignore insignificant components with low eigenvalues if necessary. After eliminating insignificant principal components, we can then form the feature vector which is a matrix of the eigenvectors we want to keep.

Deriving the new data set. The final step of Principal Component analysis is to apply our feature vector to the adjusted input data where the mean has been subtracted. The input data is transposed to get the data items in the columns and the dimensions in the rows.

$$FinalData = FeatureVector \times AdjustedInputData^{\top}$$

This gives us the original data expressed in terms of the patterns identified.

3.1.2 Generative model PCA is also a generative model. With a means-image, you can multiple the principal components with various weights to see what effect the principal component has.

4 Technical Solution

4.1 Set-up

The web-cam we have used to gather our data with is a Genius iSlim 321R that can take both regular and infra-red images. Mounted to the web-cam is a plastic frame with 4 LED lights and 4 infra-red lights that are placed just besides the LED lights. The LED and infra-red lights are controlled by Phidgets. The LED lights have colours and are placed in the following pattern: Yellow-Red-Camera-Green-Yellow. The web-cam set up can be seen on figure 1.

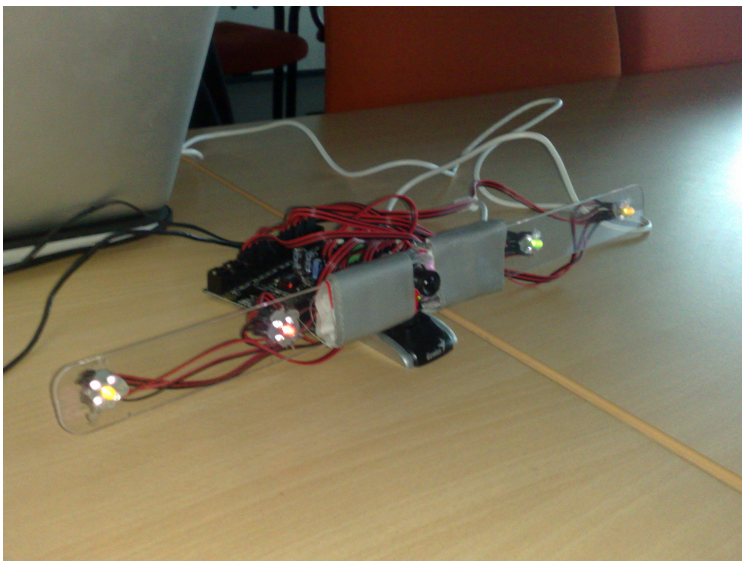

Fig. 1. Web-cam with mounted LED and infra-red lights.

Our project is written in python 2.7 using the following libraries:

- Scitkit-learn for machine learning algorithms.
- OpenCV for image analysis and manipulation.
- Numpy for matrix manipulation, also required by some of the other libraries.
- Matplotlib for plotting results.
- Phidgets python API to control the LED and infra-red lights.

To control the mounted lights, we modified a phidgets script by Adam Stelmack. See LED-simple.py for details.

4.2 Experiment

To perform the actual experiment, first of all data must be gathered. The initial scenarios we wanted to test were combinations of the head being fixated and

looking at a point, the head moving while looking at a point, and the infrared lights being turned on or off. This leaves us with the 4 scenarios with-light-head-move (WLHM), no-lights-head-move (NLHM), with-lights-head-still (WLHS) and no-lights-head-still (NLHS). Test-subjects were then asked to place their head at a set point and look at the different points see figures 2, 3, 4, 5 and 6. We then recorded video sequences of about 5 seconds in length at 30 frames per second of the various scenarios and saved the videos with an appropriate name according to the scenario.

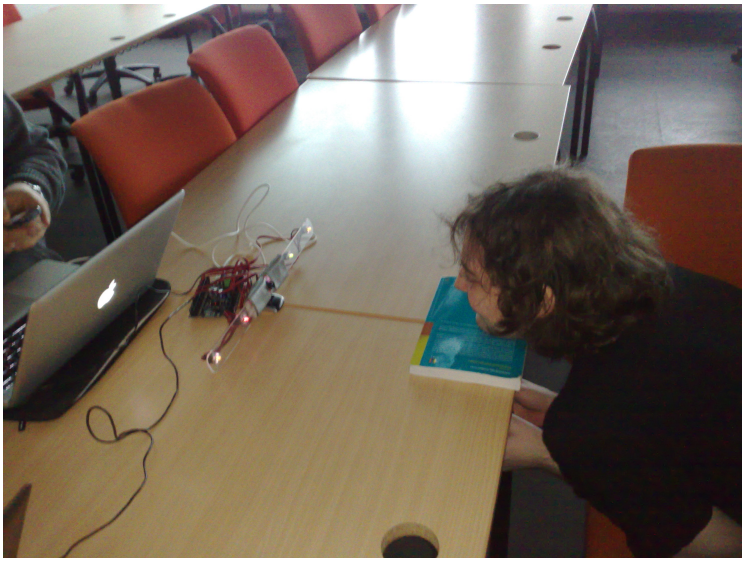

Fig. 2. Data gathering on test-subject.

Images were then extracted from the videos every 10 frames and named in such a way it was easy to identify the scenario and the frame it was from. OpenCV was used to extract the images. See Image-grabber.py for details. Bad images such as frames were the eyes were obscured or closed were then manually filtered out and added to an ignore-file.

The corners of the eye and the centre of the pupil were then marked manually using OpenCV and the three points were saved to files. See EyeMarker.py for details. This data can be used for feature based learning, though more information such as glints may be desirable.

For high-dimensional learning, we also needed to separate the eye images. To do this, eye images were extracted from the larger images using the previously marked points. To extract the image, the images were loaded and converted to grayscale. They were then histogram equalized to broaden the contrast in the image. The eye was then sliced out of the larger image and rescaled to a desired

Fig. 3. Data gathering on test-subject.

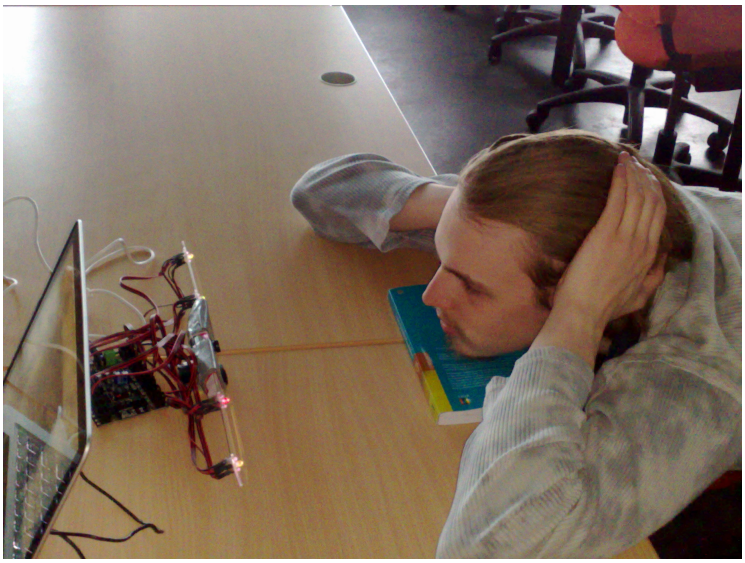

Fig. 4. Data gathering on test-subject.

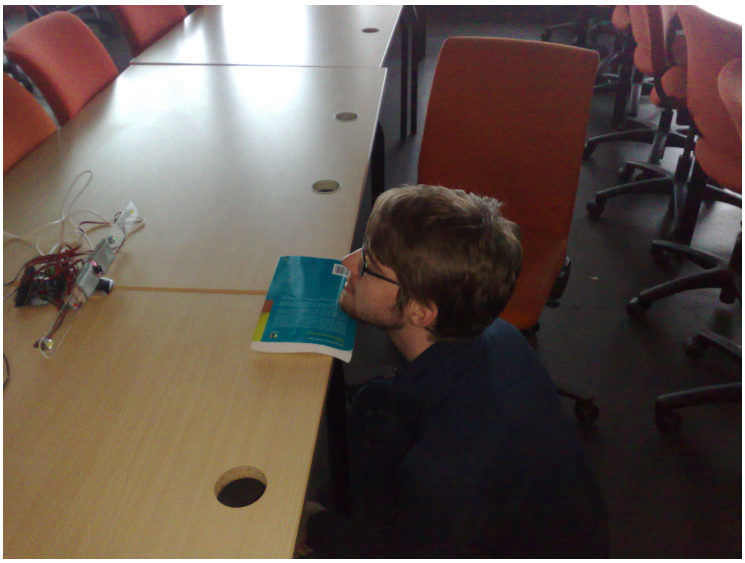

Fig. 5. Data gathering on test-subject.

Fig. 6. Data gathering on test-subject.

size, such as 20×20, 40×40 or 60×60, and then saved. For more information see TransformEyeImages.py.

PCA was then run on the eye images and the results were plotted. See run_pca.py and pca.py for more information.

After plotting the results, we identified which principal components made the different points separable and in which cases. We then tried generating images with different weights on the principal component that made the different points separable to see what effect it had on the image.

5 Evaluation

Evaluation goes here.

6 Discussion

6.1 Internal

Learning only tested on eyes from 2 people. Eye corners and pupil center was manually marked, not all marked 100% accurately. Webcam images were not of very high quality. This makes the potential usability cheaper which is a plus, but some detail may be lost. Our test setup required the test-subjects to sit a certain distance from the camera for the infrared lights to be visible. This had the effect that the eye were a smaller part of the input images than they could have been. To simplify learning, images were scaled down to 20x20 pixels. Information lost that could potentially have an impact on learning. Trade-off. Noise due to loose experiment setup. Even during head-still images the heads moved slightly etc.

6.2 External

Other ethnicities (in particular asians?) could potentially give different results. Given proper test-setup, it is unlikely to have any significant effect. "individuality of the eyes, variability in shape, scale, location, and lighting conditions"

7 Related Work

Related work goes here.

8 Conclusion

No lights head move - Not separable No lights head still - Left/right separable, PCA 1-2, 2-3, 2-4, 2-5 With lights head move - Not separable With lights head still - Separable, PCA 1-2, 2-3, (2-4), ((2-5))

References

1. Smith, L.: A tutorial on principal components analysis. Cornell University, USA **51** (2002) 52