Eye Tracking - A1

Theo Martin Meyer, Sruthi Pasham, Marjahan Begum

1 Pupil and glint detection

This section is about detecting pupils and glints in images of eyes. The implementation details and results are discussed below.

1.1 Pupil Detection

In order to detect pupils, we first needed to change the color image to Gray scale image using cv2.cvtColor. The next task was to find pupil and for that

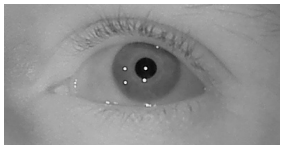

Figure 1: Grey scale of the eye

we need to identify all BLOBS[1] in the image. We used threshold value of 40 and we get the following image. While experimenting we noticed that when we have value of 200 *fitEllipse* does not work. Perhaps this is because the BLOB produce does not fit. Thresholding converts the image to 0 and 1. If value is below the threshold value it sets to 0(black) and if the value is greater than the threshold value it sets to 1(white).

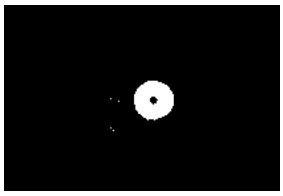

Figure 2: Applying threshold value of 40

```
thres_val = 40
ret, thres = cv2.threshold(bw_im, thres_val, 255, cv2.THRESH_BINARY_INV)
```

The next task is to identify the candidate BLOB which represents the pupil area we are interested. In order to do this we need to remove all the invalid contour. This is done by using the the following code from OpenCV. Contours are the boundaries of an objects. The boundaries are the continuous lines for

the respective objects. In this assignment they are used for detection of the pupil. This done by the cv2.findContours method from OpenCV. This function returns a python list of all the contours in the image.

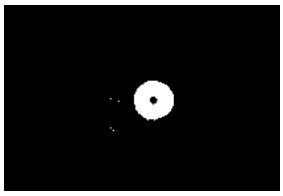

Figure 3: Applying threshold value of 40

```
conts, hierarchy = cv2.findContours(thres, cv2.RETR_TREE, cv2.CHAIN_APPROX_S

pupil = max(conts, key=lambda c: cv2.contourArea(c))
```

Given the knowledge we have the pupil, the area around a contour that is maximum will be the pupil. Hence we use the python max funtion on OpenCV contourArea as shown above. With the winning contour we fit the cv2.fitEllipse.

1.2 Glint Detection

In this section we follow the same procedure as the above up until finding contours of a glint. Then we sort the saved contours based on the distance from centre of the pupil, in increasing order and we take the first four closest points as our candidate glints.

1.3 Theory: Finding contours from binary image

In a binary image, there are only two possible pixel values, usually 0 value pixels are interpreted as background whereas 1 pixel values as foreground. Finding contours from a binary image is closely related to extracting BLOBs from the image which means analysing connectivity of the pixels i.e which pixels are neighbours and which are not (it can be 4- or 8-connectivity). There are different algorithms to find BLOBS and one of them is for example by using the Grass-Fire Algorithm. It scans the entire image from left to right and top to bottom and when it encounters a white pixel (object pixel) it labels it and "burns" it by setting it to 0(black), so that it won't be a part of yet another fire and it is not investigated again. Then it moves on to the 4 neighbours of the object pixel (or 8, depending on the type of connectivity chosen) and performs the same operations, to find out if the neighbour pixels are also object pixels, i.e. they are connected, or not. When the algorithm is done, the image shows the BLOBs, each with their own label 1 . From here, we can use a method such as findContours from OpenCV to find the curve around each BLOB.

1.4 Evaluation

Now that we have completed the detector we will evaluate how well the detector predicts.

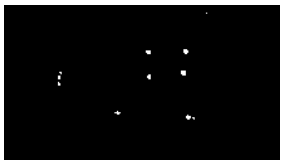

Figure 4: Applying threshold value of 210

1.4.1 Calculation for error metrics - Pupils

In order to find the difference between our detector and actual pupil we are using the difference based on the centre of the pupil.

1.4.2 Calculation for error metrics - Glints

1.5 Error Analysis

The Figures 5 and 6 shows the median and means of the errors of the pupil and glints.

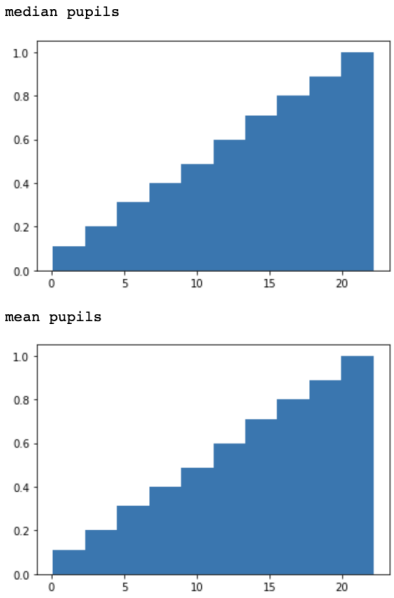

Figure 5: Median and mean of pupils errors

What the histograms suggests that almost 80% of the cases difference in the error for the pupils is below 15 pixels. Where as for the glints the errors are much lager. 80% of the cases difference in the error for the glints are between 35 and 40. This suggest glints detection is much more difficult than pupil detection. Perhaps the reason glints are bit more difficult because threshold function could not distinguish multiple reflections effectively.

This leads us to look at images that been correctly estimated and ones there are higher errors. In Figure 6 it shows correct while in Figures 7,8, 9 errors are mostly associated with glints.

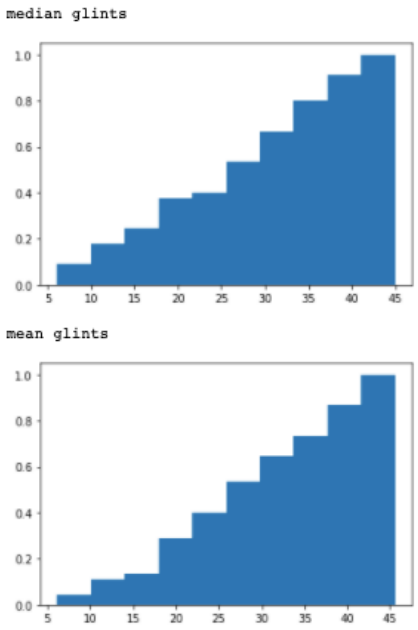

Figure 6: Median and mean of glints errors

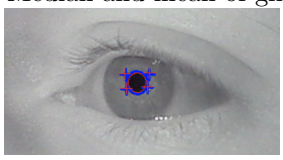

Figure 7: Both pupil and glint correct(approx)

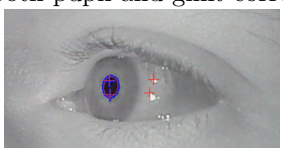

Figure 8: Correct pupil and glint incorrect(approx)

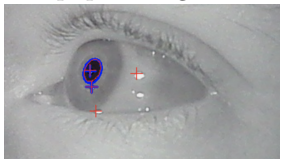

Figure 9: Correct pupil and glint incorrect(approx)

1.6 Additional Error Analysis

We tried to calculate Intersection Over Union(IOU). There were two options with it. One with the actual ground truth bounded boxes and predicted bounded boxes. We thought we would get the actual ground truth by fitting a rectangle and then fit a rectangle on the predicted pupil. We would would be able to fit a

rectangle on the predicted pupil but we did not know how to do the same with actual pupil using the jason file.

Another error analysis we wanted to carry out was to have a threshold of e.g 10 pixel. Then we identify the four values of glint difference and see if they are within the this threshold.

These two attempts are exemplified in the detector_test.ipynb. Unfortunately we could not get them to work.

2 Gaze estimation

2.1 Implementation

In the file gaze.py we are provided with a stub class GazeModel, and the task is to implement calibrate and estimate function, to be able to calibrate and estimate the screen coordinates. GazeModel gets initiated with two arguments

- calibration_images containing paths to images.
- calibration_positions containing the gaze positions.

Below are brief implementation details First we populate two design matrices Dx and Dy with the pupil centers, which our detector finds. For each image

1. We iterate through all our images contained in self.images
2. For each image, the pupil in is detected using the method find_pupil.
3. We then take the coordinates of the centre of pupil
4. These coordinates are stacked using vstack, to form a design matrix

Secondly we create the linear regression model:

1. We import the LinearRegression model from the sklearn.linear_model library
2. We use the LinearRegression 's fit method to match the pupil center we detected stored in Dx and Dy to the gaze position we have from the position.json file.
3. The two linear regression models for the Dx and Dy data are stored in the object self.lrx and self.lry respectively.

Taking about the estimate method, first the pupil coordinate in the given image is detected using the method find_pupil.Then we use the predict method from the LinearRegression objects self.lrx and self.lry with the previously detected pupil coordinates to predict the gaze position.

2.2 Evaluation

Gaze_test.py is created to test calibrate and estimate methods written in gaze.py and evaluate gaze estimation, and consists of code with below functionality.

for each folder in . . . /inputs/images:

1. We first load positions.json files consisting of ground truth values for gaze position
2. We load pupils.josn file consisting of ground truth values for pupil centres.
3. Then load images
4. With a train_size of 9 images, we create and calibrate a gaze model with the first 9 images of that folder.
5. Now that we have created and calibrated the model, next step is to estimate using the model, So , for every non-calibration image:
   - The gaze coordinates are estimated using the above model.
   - The distance between the estimated gaze position and the ground-truth gaze position is calculated.
   - The pupil is detected by using the find_pupil method.
   - The distance between the center of the ground-truth pupil and the center of the detected one is calculated.
   - These distances are saved in the corresponding arrays. (with movement and without movement)
6. The mean gaze error in the set of images without movement and with movement are calculated using numpy.mean
7. The median gaze error in the set of images without movement and with movement are calculated using numpy.median
8. A histogram plot for the gaze errors in the set of images without head movement using bins, cumulative=True, density=True is produced and saved as see in Figure 10.

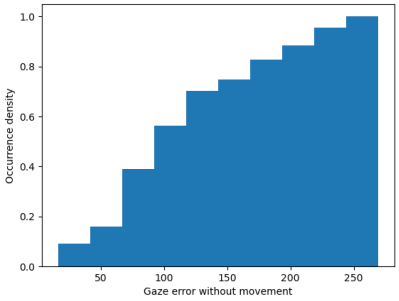

Figure 10: With out movement

9. A histogram plot for the gaze errors in the set of images with head movement using bins, cumulative=True, density=True is produced and saved as see in Figure 11.

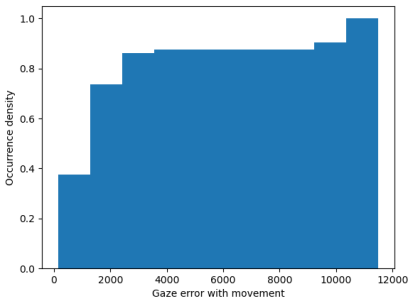

Figure 11: With movement

10. The correlation between pupil detection distance and gaze distance errors is calculated using numpy.corrcoef as shown in Figure 12.

```
correlation coeff, without movement [[ 1.         -0.17315899]
 [-0.17315899  1.        ]]
correlation coeff, with movement [[1.         0.96305735]
 [0.96305735 1.        ]]
```

Figure 12: Correlation coefficient

Below are additional metrics that could be used in evaluation.

- Precision : (TP/(TP+FP)) It is used to measure the correct predictions. TP = True postive , FP = False postive , FN = False negative.
- Recall :(TP/(TP+FN)) It is used to calculate the true predictions from all correctly predicted data.
- Intersection over union(IOU) IOU is a metric that finds the difference between ground truth annotations and predicted bounding boxes.
- Average precision(AP) To evaluate the detection commonly we use precision recall curve but average precision gives the numerical values it is easy to compare the performance with other models. Based on the precision-recall curve AP it summarises the weighted mean of precisions for each threshold with the increase in recall. Average precision is calculated for each object.
- Mean Average Precision(mAP) Mean average precision is an extension of Average precision. In Average precision, we only calculate individual objects but in mAP, it gives the precision for the entire model. To find the percentage correct predictions in the model we are using mAP.

Below is an example of good result given by gaze estimator shown in Figure 13. On the left we have the picture of the gaze and on the right we have

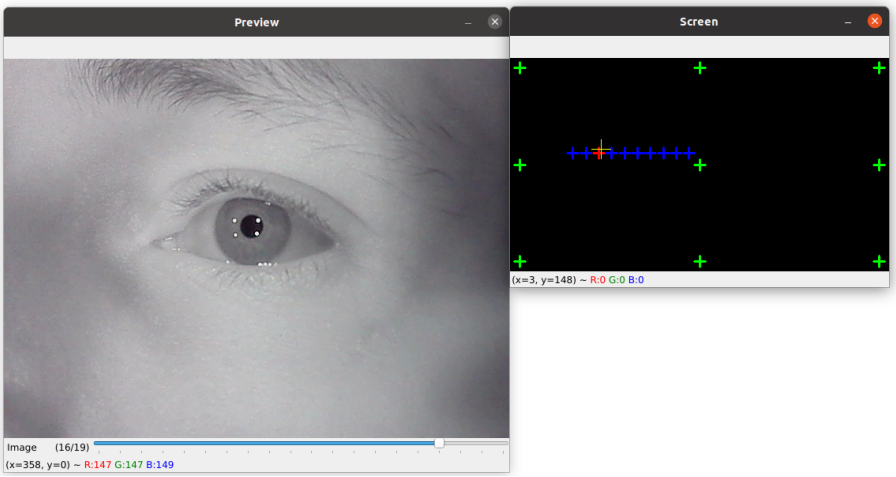

Figure 13: Good result from gaze estimator

the corresponding screen. The red cross is the true position where the gaze is pointing, and the yellow cross represent the one we estimated.

Below is an example of bad result given by gaze esimator shown in Figure 14.

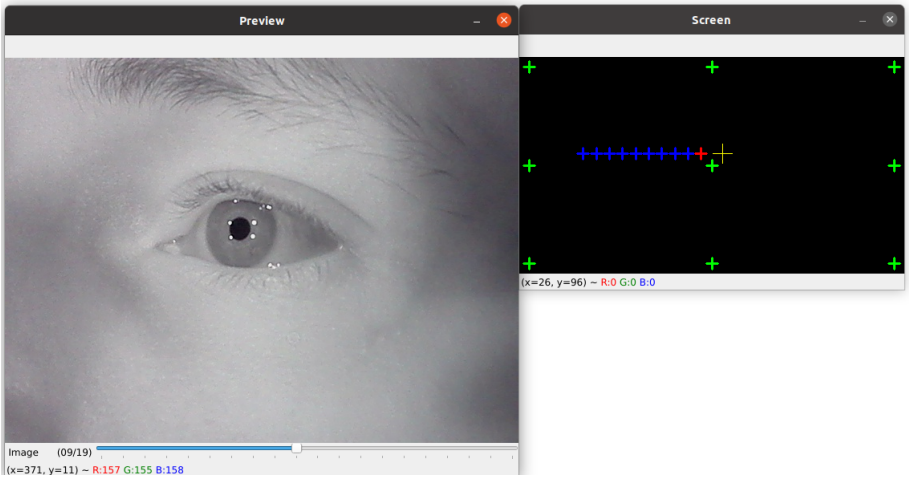

Figure 14: Bad result from gaze estimator

As can be seen from picture below, the mean errror without head movement is 121.07 significanlty lower when compared to 2679.17 from the mean error with head movement shown in Figure 15.

Also median error without movement is significantly low 108.7 when compared to 1515.16 from median error with head movement as seen below in Figure 16.

The reason could be that with head movement there is a significant change in coordinates for the pupil, and model doesn't get a chance to train well since

```
mean error without movement: 121.07981294541158
mean error with movement: 2679.175738138181
```

Figure 15: Mean error

```
median error without movement: 108.71441680647897
median error with movement: 1515.161656201048
```

Figure 16: Median error

the eye position is changing constantly due to the movement, and at the same time our model is trained with very less training data. We use only 9 images to train our model, before doing prediction. So the solution could be to have more training data, in order for the model to learn and train itself better, to be able to give good predictions.

Also it would be interesting to use ground truth values for pupil rather than using detector values during estimation, to see if the accuracy is improved for gaze estimation.

3 Improving the model with Polynomial Gaze detection

To improve our model and predict better result, we are improving our Gaze model from a simple linear regression to a polynomial regression. Polynomial regression is still a linear regression in the end, but where the feature have been combined. In our case, our dataset X has only two features: the x and y of the pupil position. Our linear regression model f looks like this:

$$f(w, X) = w_0 + w_1 x + w_2 y$$

With polynomial regression, we are using a linear regression, we our going to use combination of our two features x and y, for example if we use a polynomial regression of order 2, we will use a linear regression of 5 features instead of 2:

$$f(w, X) = w_0 + w_1 x + w_2 y + w_3 xy + w_4 x^2 + w_5 y^2$$

A polynomial regression of order 3 would be a linear regression of 10 features:

$$f(w, X) = w_0 + w_1 x + w_2 y + w_3 xy + w_4 x^2 + w_5 y^2 + w_6 x^2 y + w_7 xy^2 + w_8 x^3 + w_9 y^3$$

Using polynomial regression can improve prediction if the data doesn't follow a straight line or plane model.

From our tests we used the regular linear regression from the previous exercise, and then we added result from a polynomial regression of different orders (2, 3 and 4).

We also used training datasets and tests datasets of different sizes.

The metric we use to compare the different models is sum of the distances in pixel between the real position of the pupil and the one we predicted. We

also used the average error in pixel between the real position and the detected one.

We used pattern0, pattern1 and pattern2 as training data for our models, and pattern3 as test.

Model	Sum	Average
linear regression	4407	231
polynomial regression order 2	4279	225
polynomial regression order 3	4746	249
polynomial regression order 4	4910	258

The result show that a polynomial regression of order 2 give the bests result. The result are not improving as much as we though. We went from 231 pixel errors to 225

After that, it seems that the model overfit the data and fail to deliver a better result than the simple linear regression. We tested with orders up to 19, and the result are getting worse as the order increase, but not in a linear way.

Model	Sum	Average
linear regression	4407	231
polynomial regression order 2	4279	225
polynomial regression order 3	4746	249
polynomial regression order 4	4910	258
polynomial regression order 5	4587	241
polynomial regression order 6	4716	248
polynomial regression order 7	4599	242
polynomial regression order 8	5368	282
polynomial regression order 9	8449	444
polynomial regression order 10	10810	568
polynomial regression order 11	9903	521
polynomial regression order 12	11684	614
polynomial regression order 13	11483	604
polynomial regression order 14	8876	467
polynomial regression order 15	8617	453
polynomial regression order 16	8996	473
polynomial regression order 17	9170	482
polynomial regression order 18	7411	390
polynomial regression order 19	7805	410

225 pixels of error in average seems high for a 1030 by 1870 image. We made the hypothesis that this is due to the test dataset (pattern 3) being too different from the training dataset (pattern 0, 1 and 2). To resolve this issue, we decided to create a new training dataset with the 4 pattern, and then randomly extract 19 (the size of pattern3) images from the training dataset to the test dataset.

Model	Average error
linear regression	173
polynomial regression order 2	163
polynomial regression order 3	155
polynomial regression order 4	158
polynomial regression order 5	161
polynomial regression order 6	162
polynomial regression order 7	166
polynomial regression order 8	167
polynomial regression order 9	236
polynomial regression order 10	212
polynomial regression order 11	306
polynomial regression order 12	208
polynomial regression order 13	202
polynomial regression order 14	259
polynomial regression order 15	250
polynomial regression order 16	250
polynomial regression order 17	252
polynomial regression order 18	258
polynomial regression order 19	254

Using random sampling for the test dataset we see that the average error drop at 155 pixels, we also see the polynomial regression of order 3, 4, 5 and 6 give better result than the polynomial regression of order 2. Using random sampling improved the result by almost 30%.

We also run tests on the on the moving dataset (moving medium and moving hard), using random sample (size = 19) to separate the training and tests datasets. We got unprecise results as expected, and the use of polynomial regression doesn't seems to improve the result significantly.

Model	Average error
linear regression	667
polynomial regression order 2	692
polynomial regression order 3	626
polynomial regression order 4	597
polynomial regression order 5	644
polynomial regression order 6	652
polynomial regression order 7	856
polynomial regression order 8	868
polynomial regression order 9	742
polynomial regression order 10	1065
polynomial regression order 11	1133
polynomial regression order 12	1076
polynomial regression order 13	1220
polynomial regression order 14	1241
polynomial regression order 15	1538
polynomial regression order 16	1372
polynomial regression order 17	1784
polynomial regression order 18	1475
polynomial regression order 19	1483

References

[1] Paulsen, R. R., Moeslund, T. B. (2011). Introduction to Medical Image Analysis. DTU Informatics, Technical university of Denmark. https://vbn.aau.dk/en/publications/introduction-to-medical-image-analysis