Outstanding Academic Papers by Students

學生優秀作品
MOOC-Gaze: Online Solution for Tracking Learners’ Gazing Dynamics of MOOC Videos

by

ZHOU XINGHAI, Student No: DB325062

Final Project Report submitted in partial fulfilment of the requirements of the Degree of Bachelor of Science in Computer Science

Project Supervisor

Dr. Shirley Weng In Siu

26 May 2017

Vol. 1
DECLARATION

I sincerely declare that:

1. I and my teammates are the sole authors of this report,
2. All the information contained in this report is certain and correct to the best of my knowledge,
3. I declare that the thesis here submitted is original except for the source materials explicitly acknowledged and that this thesis or parts of this thesis have not been previously submitted for the same degree or for a different degree, and
4. I also acknowledge that I am aware of the Rules on Handling Student Academic Dishonesty and the Regulations of the Student Discipline of the University of Macau.

Signature : ____________________
Name : ZHOU XINGHAI
Student No. : DB325062
Date : 26 May 2017
ACKNOWLEDGEMENTS

The author would like to express his utmost gratitude to UM for providing the opportunity to carry out a project as a partial fulfilment of the requirement for the degree of Bachelor of Science.

Throughout this project, the author was very fortunate to receive the guidance and encouragement from his supervisor Dr. Shirley, Weng In Siu and Mr. Miguel Gomes da Costa Junior.
ABSTRACT

Video-based lecturing has become the major teaching-and-learning activity of Massive Open Online Courses (MOOC). While students can learn any time at their own pace, course providers cannot have face-to-face communication with students to receive instantaneous feedback. Conventional ways like web forums allow learners to submit feedback after class, but this demands their additional effort to reflect on the lecturing process. Therefore, new techniques to evaluate learners’ in-class behaviour are needed. Gaze tracking is a technique which keeps track of the gaze position of a user. As a standard computer comes with a web camera, it is practical to perform non-intrusive gaze tracking of MOOC learners directly from user’s image. In this project, we created MOOC-Gaze, the software application to predict users' gaze positions of a MOOC video by analysing user’s facial and eye dynamics via web cameras. The prediction model is essentially a ridge-regression trained mapping function which takes users’ facial picture as input and outputs predicted gaze position as screen coordinates. The prediction model is built on-the-fly and user-specific, which requires one short calibration procedure before user starts watching the video. The predicted gaze positions can be visualized in heat-map form in a video replay for educators to observe learners’ individual and aggregated gaze patterns. This post-watch analysis can help in course evaluation, e.g. to find out whether gazing positions matching the expectation of the designer of the course video. MOOC-Gaze was designed to work with any kinds of video and to operate in a normal web browser. To improve the prediction accuracy, a literature survey of gaze tracking techniques was conducted and some solutions were proposed and tested. Our web framework together with the improved gaze prediction method delivers a new online gaze-tracking solution that can also be used in use cases other than MOOC.
# Table of Contents

CHAPTER 1. INTRODUCTION................................................................. 10  
1.1 Motivation.................................................................................. 10  
1.2 System Overview ...................................................................... 11  
1.3 Contributions........................................................................... 13  

CHAPTER 2. LITERATURE SURVEY............................................... 14  
2.1 Gaze Prediction........................................................................ 14  
2.1.1 Model-Based Methods.......................................................... 14  
2.1.2 Appearance-Based Methods................................................ 15  
2.2 Gaze Feedback Analysis......................................................... 17  

CHAPTER 3. METHODS................................................................. 18  
3.1 Image-Based Gaze Prediction.................................................. 18  
3.2 Eyes Image Extraction.............................................................. 19  
3.2.1 Identifying user’s face as a bounding box............................. 19  
3.2.2 Extracting eye images from a bounded face area................ 21  
3.3 Prediction Model...................................................................... 22  
3.3.1 Theory of Ridge Regression................................................ 22  
3.3.2 Applying ridge regression in gaze prediction..................... 23  

CHAPTER 4. SYSTEM DESIGN....................................................... 25  
4.1 Functional Specifications.......................................................... 25  
4.1.1 Functional Requirements.................................................... 25  
4.1.2 Non-functional Requirements............................................. 25  
4.2 System architecture................................................................... 25  
4.3 Primary Work Flow................................................................. 26  
4.4 Conceptual Class Diagram....................................................... 27  
4.5 System Components Diagram................................................ 27  
4.6 Sequence Diagram.................................................................... 28  

CHAPTER 5. IMPLEMENTATION.................................................. 30  
5.1 Implementing the gaze-tracking solution................................. 30  
5.1.1 Problem of ClmGaze.............................................................. 30  
5.1.2 Improvement: Customize jsFeatGaze................................... 30
LIST OF FIGURES

Figure 1: an eye tracking device in the market .................................................. 11
Figure 2: The MOOC-Gaze System Concept .................................................. 12
Figure 3: Image-based gaze prediction proposed in this study .......................... 18
Figure 4: feature prototypes (patterns) ......................................................... 19
Figure 5: Cascading classifiers ........................................................................ 21
Figure 6: Sample source image for eyes extraction ....................................... 21
Figure 7: Second strip of a bounded face ...................................................... 21
Figure 8: Method to extract two eye images ................................................ 22
Figure 9: Resulted image after eyes locating ................................................ 22
Figure 10: Client-Server Architecture ......................................................... 25
Figure 11: Primary Flow Chart ....................................................................... 26
Figure 12: Conceptual Class Diagram ............................................................. 27
Figure 13: System Components Diagram ...................................................... 28
Figure 14: Main Sequence Diagram ............................................................... 29
Figure 15: Eye Extractor Interface ................................................................. 31
Figure 16: jsfeat face detection demo: video-canvas logic, page statistics tool and detection logic crowded in the same namespace with shared variables ......................... 31
Figure 17: one line detector for flexible use after improving jsfeat demo code ....... 32
Figure 18: Using Promise.then() to order asynchronous loading of scripts ....... 33
Figure 19: Scaling videos to take maximum area ......................................... 34
Figure 20: State diagram of HeatmapPlayer .................................................. 35
Figure 21: central event bus ........................................................................... 35
Figure 22: sample usage 1 of event bus ...................................................... 36
Figure 23: Publish course form - filled by course providers ......................... 37
Figure 24: Landing page - list of courses .................................................... 37
Figure 25: Calibration sample: user has to click the blue rectangle ............... 38
Figure 26: Calibration sample: user has to click the blue rectangle .................................................. 38
Figure 27: Calibration sample: user has to click the blue rectangle .................................................. 39
Figure 28: Calibration sample: user has to click the blue rectangle .................................................. 39
Figure 29: user watching a course video .......................................................................................... 40
Figure 30: Replay - Sharpen color represents more gazes at the same frame ............................... 40
Figure 31: Replay - Sharpen color represents more gazes at the same frame ............................. 41
Figure 32: Eye extraction test results of 450 human face images by jsFeatGaze. Cases with successful eyes recognition are indicated with a blue rectangle at the top left corner of the image ................................................................. 43
Figure 33: continued ......................................................................................................................... 44
Figure 34: continued ......................................................................................................................... 45
Figure 35: Eye extraction test results of 450 human face images by ClmGaze. Cases with successful eyes recognition are indicated with a blue rectangle at the top left corner of the image ................................................................. 46
Figure 36: continued ......................................................................................................................... 47
Figure 37: continued ......................................................................................................................... 48
Figure 38: Test user image sample .................................................................................................... 49
Figure 39: Distributions of user clicks on the screen in one 50-point calibration cycle, .......................... 50
Figure 40: Line plot of training errors of jsFeatGaze and ClmGaze for large training set size. ................................................................. 51
Figure 41: Box plot of training errors of jsFeatGaze and ClmGaze for large training set size. .................. 52
Figure 42: Line plot of testing errors of jsFeatGaze and ClmGaze for large training set size. ................................................................. 53
Figure 43 Box plot of testing errors of jsFeatGaze and ClmGaze for large training set size. .................. 53
Figure 44: Line plot of training errors of jsFeatGaze and ClmGaze for small training set size. ................................................................. 54
Figure 45: Box plot of training errors of jsFeatGaze and ClmGaze for small training set size. jsFeatGaze’s errors are very small and close to 0 line ................................................................. 55
Figure 46: Line plot of testing errors of jsFeatGaze and ClmGaze for small training set size. ................................................................. 56
Figure 47: Box plot of testing errors of jsFeatGaze and ClmGaze for small training set size.
CHAPTER 1. INTRODUCTION

1.1 Motivation

With the rise of MOOC (Massive Open Online Course) in education, learning via videos gradually gains its popularity. As the transition of live spot teaching, it’s important to have adequate techniques to evaluate students’ feedback towards these pre-recorded materials, e.g. their attentiveness distribution over the video duration, how much they understand on each part of the video lecture, etc.

Web forum has been a very general used platform to collect students’ feedback, which is used by many MOOC platforms like Coursera. However, this method applies only to those who are willing to ask questions or give answers, which may not be really helpful to relatively passive learners who only watch a video and leave. There have been studies analysing students’ behaviour in video learning by tracking user clickstreams on video-playing manipulation, e.g. fast forward, rewind, pause, and so on [1]. These natural actions for a video watcher may tell how a MOOC learner spend his focus on different sections of a video, his extent of engagement in the course material and even a prediction of his continuity on this web course [1]. The advantage of this method is that it does not require students’ additional effort to provide feedback nor to be interrupted while studying.

Inspired by click stream method, this project aims to compose another type of non-interruptive solution that helps capture MOOC learners’ reaction, which is by gaze prediction techniques, provided that web-cameras have been so widely spread in personal devices nowadays. Hence this project may contribute to MOOC learner behavioural studies together with the clickstream method mentioned above.

In addition, gaze prediction through a normal web camera also contributes to the evaluation of MOOC teaching and learning in many ways, e.g. the gazing path on a course slide of a student can become a useful feedback for the course material maker. The common gazing area in the video of a large group of students may reveal their common interests, the duration change of their gazing at the same area may tell information about their study performance, for example, a student may spend less gaze time on a formula which he has learnt in the past than another student that has no background knowledge about that formula.

This technique is promising to facilitate remote studies. In fact, Gaze prediction is a long-discussed topic in human-machine interaction. This refers to the technique which predict the exact region of a screen that a user is currently looking at, given his face/eyes image captured by camera or a specific device. This project mainly focuses on the solution that does not require special device but only a general web-camera, which is supposed to be applied in MOOC classrooms. The project aims to propose a practical solution, with implementation open-sourced to facilitate the public adoption of this technique. Although this project is inspired and designed mainly for MOOC use cases, other scenarios may also benefit from it, e.g. assistant human-computer communication for disabilities, live streaming app user interaction enhancements, and so on.
We built MOOC-Gaze, a special MOOC platform which can record and predict users’ gaze positions via a normal web camera while they are watching a course video. The gazing dynamics will be analysed and feedback to course provider and student in report form to aid understanding about the class learning performance.

1.2 System Overview

This project aimed to build a web-based platform where course providers can put their resources and opened to the public. In addition to such common functionalities which current MOOC platforms already have, the project tried to incorporate a gaze tracking solution into platform to track the gazes of users while they are watching the course videos. The recorded gazes can be later visualized as a heatmap overlaid on each frame of a corresponding video’s replay. Course providers can also view the statistics of users’ gaze data on the platform. The functions serve as a way of feedback to help course providers better understand how the course video influence their online students’ behaviour by observing their gaze dynamics.

Below is an overview of how the platform operates to support course providers and students in MOOC learning.
Figure 2: The MOOC-Gaze System Concept
1.3 Contributions

In this project, we have the following major achievements:

- Applied gaze-tracking techniques in MOOC education field to provide a new way of student feedback, which the feedback data is not questionnaires, not forums, but gazes
- Implemented MOOC-Gaze platform, the first MOOC platform which can track users’ gazes via a normal web camera
- Applied heatmap technique as gazed position visualization in MOOC videos
- Improved the performance of a recent web-based gaze tracking package
CHAPTER 2. LITERATURE SURVEY

In order to track learners’ gaze dynamics and provide useful feedback to course providers, there are two main challenges we have encountered: gaze tracking and gaze feedback analysis. Gaze tracking provides the position in the video which learners are focusing on. With these positions, gaze feedback analysis provides useful feedback to course providers. In this chapter, we will summarize the related work in these two fields.

2.1 Gaze Prediction

In the literature, it is almost well-recognized that the approach towards gaze prediction is divided into two sets: model-based methods and appearance based methods.

2.1.1 Model-Based Methods

In model-based methods, features of face, head pose and eyes are carefully designed and extracted from a user-capture image with different kinds of image processing techniques. These features are extracted based on a carefully designed 2d/3d features model, e.g. a 2d eye model composed of eye lids and eye centres as features [3], a 3d head model including facial features and facial contour symmetry information as features [4]. These features are then map to screen coordinates/shifted angles compared to anchor position, by a special model/function, e.g. linear regression model, support vector machine trained model, or simply manually crafted formula-based model. Usually the work flow in a model-based solution goes through the following steps: facial feature extraction, eye-specific feature extraction, building mapping function and output gaze position. Precise eye-region localization in an image and the mapping function are the two critical steps in a gaze prediction solution.

Recently Sckodras proposed a feature-based solution which predict horizontal gaze direction and vertical gaze direction by two different mapping functions respectively, where eye centers and eyelids are key features of the prediction model [3]. In Skodras's solution, a user's image captured by the camera is firstly processed with a face detection algorithm proposed by Viola and Jones [5]. Then, a heuristic approach is adopted to locate the rough region of eyes. The regions of eye centres are then identified by a method which utilize symmetrical characteristics and chrominance information of the rough regions. After having the exact pixels of eye centre, it continues to locate the upward eye lid and downward eye lid, which is latter used for determining the openness of the eyes. The solution also identifies a block of pixels, which is an image patch containing eye border and eye brown border. The centre of the image patch is the reference point to calculate the displacement of eye centres and eye lids in frame sequences. Given the displacement distances and known gaze points in calibration phrase, two linear regression models are trained to predict horizontal and vertical gaze directions respectively. The displacement features of eye centres are used in horizontal model while the displacement features of both eye centres and eye lids are used in vertical model. The calibration duration of this solution lasts for a few seconds for a user. The solution obtains a less than 2-degree error in testing.

Lin's proposed solution particularly addresses gaze prediction obstacles caused by illuminance variety of user environments [6]. After detecting the face with Viola-Jones's method [7], it processes the image with its proposed light filtering technique,
including gray-scale process and some customized color filtering functions, which turns out to be effective in removing the lighting noise in its experiments. After light filtering, the face image pixels are grouped into components by pixel connectivity as candidates for eye regions. The eye regions are selected based on a synergy of heuristic rules, symmetry characteristics and the relative distance between an eye and the face center. After locating the eye regions, pixels of irises are found by the following method: selecting pixels that is under 10% of Y value of the YCbCr color space of its histogram-equalized eye regions. A feature vector composed of 40 selected Fourier descriptors of two eye regions and irises are then trained with support vector machine given known gazing directions in calibration phrase. Lin uses his own source of data, which contains over 10000 images, in which about 3/10 the subjects wear glasses. It reports 90% of eye detection accuracy and gaze prediction accuracy.

Heyman's solution purely depends on model-based mathematical calculation without the need of machine learning in its prediction step [4]. This solution takes head poses into consideration. It also starts its work by face detection with Viola-Jones algorithm [5]. It then tracks the detected facial features based on the normalized sum of squared differences (NSSD) with template matching technique, to calculate the displacement vectors of facial features in user head movements. It also uses a Bayes's classifier to locate pixels of irises. The prediction model uses head pose information to calculate a rough direction and then fine-tune the result with eyeball orientation. In details, it builds up an initial frontal face model at the stage of facial feature detection. Based on this initial model, a series of computations are carried out to get the yaw angle, pitch angle and rotational angle of the head and eyeballs with displacement vectors, i.e. to "re-project" the user face onto a "cylindrical head and spherical eyeball model." The eyeball yaw angle and eyeball pitch angle are the final output of the horizontal direction and the vertical direction of this prediction model, respectively. It reports 7.09 degree mean error at horizontal direction and 4.4 degree mean error at vertical direction prediction.

2.1.2 Appearance-Based Methods

In appearance based methods, the photometric appearance is directly used as input to predict gaze. These methods capture other objects not limited to the eyes subject. With enough training data, other environment factors can be learnt into the model, which gives the potential capability to cope with variant conditions like illumination variation, head pose, quality of images etc. These methods allow gaze prediction technologies to be used in normal unconstraint conditions.

Appearance based methods typically require larger amounts of training data than model based methods, and the result closely relies on the training dataset. In the publishes, there has been several public datasets which is related to our system:

MPIIGaze dataset, proposed by Zhang [8], contains 15 subjects with total 213,659 images. The dataset is collected in variant size of laptops. Moreover, the sample are freely distributed over different daytime over more than three months, so to simulate in-the-wild setting. Zhang also proposed algorithm based on MPIIGaze dataset, the pre-processing employs Li et al.’s SURF cascade face-detection method [9] and Baltrušaitis et al.’s constrained local mode framework to locate facial landmarks [10]. Then predicts 3D head pose by using 3D facial shape model and normalizes the image in training space to get a centralized position and fixed distance. After pre-processing, the model is trained by using multimodal convolutional neural networks (CNN). MPIIGaze
Tracking Learners’ Gazing Dynamics

dataset with the proposed algorithm can achieve accuracy of 13.9 degrees. However, unlike most of other public datasets, we found that MPIIGaze dataset contains only the normalized eyes images. In the MOOC use case, since the input images need pre-processing before predicting the gaze position by the model, the pre-processing especially 3D reasoning is not ideal to running in web browsers. Moreover, the achieved accuracy is limited.

TabletGaze Dataset proposed by Huang et al. [11] contains 51 subjects, and provides total 816 video sequences instead of images. The dataset is collected in 4 predefined postures: standing, sitting, slouching, and lying, but has no constraint for how they hold the tablet. The dataset is collected in single tablet (Samsung Galaxy Tab S 10.5). Based on Tablet Dataset, the algorithm performs the pre-processing and normalization so that the valid images can be extracted and has the comparable scale. Then eye-image cropping is performed by using a cascade eye detector [12]. Then the 5 features including contrast normalized pixel intensities, LoG, LBP, HoG, mHoG are extracted. Finally Use Random Forest with 100 trees to train the model, so to take advantage of the strong performance for feasibility in scaling to large dataset. By this method, in different environment settings, the experiment shows the mean error 3.17cm in person independent prediction, mean error of 2.5cm in person dependent prediction. However, there is no evaluation of prediction across different datasets, which bears the risk of significant dataset bias [13, 14]. Moreover, TabletGaze Dataset works in tablet settings rather than laptop settings. To work across devices, this method relies on transfer learning approach which gives difficulty to use in our project.

GazeCapture dataset proposed by Krafka et al. [14] contains 1474 subjects, 2,445,504 images, which is about 30 times as many participants as other datasets. The dataset is established by using mobile phones and tablets, which is scalable toward crowdsourcing approaches to produce large size and variability. The proposed algorithm iTacker for gaze prediction is relatively simple, it mainly uses convolutional neural networks (CNN), so to make best use of the large dataset, iTacker employs two characteristics to increase robust capability toward poor-quality eye detection: First, it does not rely on manual engineered system for head pose detection and normalization. Second, it takes two of more discriminative regions for prediction, including an image crop of the face together with its location in the image (termed face grid), as well as an image crop of tight region of the eyes. In order to support mobile devices usage with limited resources, the model complexity is reduced by lowering the image quality of other non-discriminative region, also by combining using a full model and a reduced model for prediction. The method can achieve prediction error of 1.71cm and 2.53cm without calibration on mobile phones and tablets respectively, and 1.34cm and 2.12cm with calibration on mobile phones and tablets respectively. Proposed experiment also shows it generalizes well over other datasets, with error of 2.58cm. The prediction time can be as fast as 0.05s for reduced model, with Apple face detection pipeline, overall can achieve 10-15fps. GazeCapture with iTacker overall achieve a good result. However, GazeCapture dataset is only available for permitted access at that time when our work is completed.

Although appearance based methods are believed to work well in normal unconstraint condition, and have potential to give acceptable accuracy, the several public datasets we have found neither have no permission to access nor have their own limitation which we have mentioned above, such as performance, variability over devices etc. On the
other hand, building our own dataset is costly thus impractical to our project. As a result, we tend to use model based method in our system.

2.2 Gaze Feedback Analysis

Gaze feedback are useful in many ways for evaluating teaching and learning of MOOC video, researchers are finding meaning of gaze pattern in terms of learning outcomes. Sharma et al. [15] found the relationship between gaze coverage and learning performance. In their experiment, Areas of Interests (AOIs) are predefined in every slide in the video, which are visual blocks of content in the slide. 40 participants are asked to watch the video, then finished a post-test after watching the video. The measurement of learning performance is based on the post-test result. They found that students with good learning performance have the larger gaze coverage area (measured by the number of Attention Points) while students with bad learning performance have smaller gaze coverage area. They also found that students with good learning performance have less AOI missed while students with bad learning performance have more AOI missed.
CHAPTER 3. Methods

3.1 Image-Based Gaze Prediction

In order to achieve gaze prediction with a web camera, various techniques have been composed as a combination, including machine learning and image processing. The methods explained in this section are a composition from two work: Webgazer package [16] and jsfeat computer vision library [17], where the former work proposed and implemented the prediction modelling part and the later work implemented the face detection, which our own-proposed eye extraction method depended on.

Our method focus on how to train and predict a user’s gaze position with improved accuracy over the existing method, exploiting user’s camera-captured frontal picture as input. In general, the gaze prediction can be divided into two phases: The training or calibration phase, and the testing or prediction phase. The image below illustrates the whole process including eyes extraction, model training and applied prediction.

As shown in the following figure, the key component is the trained model, which takes input from an image processing flow where eye images were extracted from a person’s frontal image, and output two dimensional coordinates that corresponds to the screen position he person gazes at. The face-eye extraction model as well as the regression model will be explained in the following paragraphs.

![Image-based gaze prediction proposed in this study](image-url)
3.2 Eyes Image Extraction

Given that in our use cases, our source image is a frontal camera-captured user image. The solution must be equipped with a method to extract eye images from a frontal-facing user picture. The extraction also consists of two consecutive process: identifying the face bound box in the image and extracting eye images from the face region.

3.2.1 Identifying user’s face as a bounding box

This method of finding a face in an image was drawn from the work of Lienhard & Maydt [18], also known as extended haar-object detection, which is an improved version of the classical face detection method proposed by Viola Jones, et al. [5]. This is a method to train a classifier which identify objects in an image, in this case, a human face, by the following sub-steps:

1. Calculating extended haar-like features of the training samples for all windows
2. Prepare a classifier for each calculated feature
3. Put classifiers into multiple groups(stages)
4. Using AdaBoost algorithm to form a strong classifier for each stage
5. Output final model: one classifier which cascades the strong classifier of each stage

3.2.1.1 Calculating extended haar-like features of the training samples for all windows

A window is a group of pixels defined by a width W and a height H. A pattern splits a window into two rectangles, r0 and r1, where r0 represents the whole window and r1 represents the white window. A list of patterns, also known as feature prototype are listed below as proposed in [18]:

![Feature prototypes](image)

Figure 4: feature prototypes (patterns)

An extended haar-like feature corresponds to one pattern, which means for one window of pixels there will be a list of features. The feature is calculated as the following formula:

\[ feature_i = w_0 \cdot Sum(r0) + w_1 \cdot Sum(r1) \]
Here $w_0$ is always set to -1 and $w_1 = \text{Area}(r0)/\text{Area}(r1)$. Sum represents the sum of all the pixels in the rectangle $r$.

### 3.2.1.2 Prepare a classifier for each calculated feature

A classifier $h_t(x)$ consists of a feature and a binary value which indicates “is” or “not” a face-contained area. The number of classifiers is equal to the number of calculated features.

### 3.2.1.3 Put classifiers into multiple groups (stages)

All classifiers will be group into different stages heuristically. For example, 250 classifiers can be group into 10 stages of which contains 10 classifiers.

### 3.2.1.4 Using AdaBoost algorithm to form a strong classifier for each stage

The AdaBoost algorithm was taken from [7] to train a strong classifier. A strong classifier $h(x)$ is a linear combination of each weighted classifier $h_t(x)$ in the stage, which is obtained by reweighting training samples for multiple rounds of training:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

*Equation 1*

$\alpha_t$ is the weight assigned. $h_t(x)$ outputs either 0 or 1 to indicates positive face portion or a negative face portion. A detail of the way of assigning weights to the feature-based classifiers can be found in [7].

### 3.2.1.5 Output final model: one classifier which cascades the strong classifier of each stage

The final model cascades all stage-strong classifiers obtained by AdaBoost algorithm. A cascaded model works in this way: each input will go through the stage-strong classifiers one by one until it is classified as negative, in which case the input will be output as negative and the classification process stops immediately. An image illustration is as the following:
3.2.2 Extracting eye images from a bounded face area

After obtaining the haar-feature face detection model, the area of eyes must be extracted. To extract two eyes from a face a simple heuristic approach is adopted, inspired by [3] and [6], which used a similar predefined proportional method to split the face into smaller sections and take two of them heuristically. In this project, a face will be evenly horizontally split into four stripes, where the second stripe is taken out. Two rectangles which contains the eyes is further extracted from the second stripe. The process is illustrated as below:

Source Image:

![Source Image](image1)

Figure 6: Sample source image for eyes extraction

Extracted Face:

![Extracted Face](image2)

Figure 7: Second strip of a bounded face

Extracted Eyes
3.3 Prediction Model

Image-based gaze prediction is essentially a regression problem. Given features of user images, specifically the eyes images, and user’s gaze position at the monitor screen, the prediction model is to establish a mapping function to calculate the gaze position directly from the features. Based on our experiments on the Webgazer program, we realized that when eyes images were correctly provided, there was a linear relationship between the image features and the gaze position. For an online program where instantaneous feedback is a prerequisite, simple but accurate prediction model is highly desirable. Therefore, for the gaze prediction, ridge regression – a linear regression model with regularization – has been selected in our method.

3.3.1 Theory of Ridge Regression

The core method of this project’s gaze tracking solution was essentially a prediction model which takes two eye images as input and outputs the gazing position as (x, y) coordinates as proposed in [16]. It uses ridge regression method as the prediction model which penalizes the size of the regression coefficients while fitting a linear model to the given data points.
The fundamental idea of the prediction is ridge regression

\[
\hat{\beta}_{\text{ridge}} = \arg\min_{\beta \in \mathbb{R}^p} \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 + \lambda \sum_{j=1}^{p} \beta_j^2
\]

*Equation 2*

This regression model consists of two parts: 1) Linear regression model, 2) coefficients penalty

### 3.3.1.1 Linear Regression

Given a set of mappings of variable $x$ and the dependent variable $y$, i.e. $(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i)$. An optimal value $\beta$, which is a vector of coefficients, can be found to minimize the accumulated difference between each pair of $x_i^T \beta$ and $y_i$.

\[
\sum_{i=1}^{n} (y_i - x_i^T \beta)^2
\]

*Equation 3*

### 3.3.1.2 Coefficients penalty

A constant lambda is chosen to penalize the linear model by timing the accumulated square of the coefficients to prevent from over fitting. This also makes the whole regression model more generalized.

\[
\lambda \sum_{j=1}^{p} \beta_j^2
\]

*Equation 4*

### 3.3.1.3 Benefits of ridge regression over simple linear regression

One of the benefit using ridge regression over linear regression is this: when an appropriate lambda is chosen, the coefficients penalty, serving as a constant base, can prevents the whole model’s value from fluctuating too much when the linear regression tends to fit the training data by add over-amount of coefficients, and therefore, increases generalization and avoid over fitting.

### 3.3.2 Applying ridge regression in gaze prediction

The input of our ridge regression model is the images of two eyes, represented by 120 pixels, where each eye has $6 \times 10$ pixels.
The output of the model is gaze position measured as screen coordinates in pixels. That means there were two values as output, i.e., x coordinate and y coordinate. Hence two ridge regression models are built for x and y coordinates prediction separately, although they take the same 120 pixels input. In other words, the model contains two set of coefficients for horizontal and vertical position prediction.

The penalty is set to $\lambda = 10^{-5}$ as heuristically proposed in [16].
CHAPTER 4. System Design

4.1 Functional Specifications

4.1.1 Functional Requirements

The system was designed to have at least the following functions:

1. Course providers can publish their video courses via the platform
2. Online students can watch the course videos online via the platform
3. While students are watching videos, the position of the video they gazed should be recorded
4. Course providers can watch course videos replay with heatmap overlaid on top to indicate the gaze position of students for evaluation

4.1.2 Non-functional Requirements

1. Students' gaze position must be able to be calculated in at most 1 second
2. Students’ frontal image captured by camera should not be transferred through the network for privacy reasons

4.2 System architecture

Our system was designed as a simple client-server architecture:
4.3 Primary Work Flow

The gaze tracking function relies on a prediction model to perform gaze prediction given user’s frontal image. In this prototype, the prediction model must be trained or calibrated for each individual at each time he watches a course, assuming that the illumination and head position vary every time. For future, the prediction model can be stored as part of the user’s profile which can be re-used for the next user’s access. Here, the work flow of that a user watch a user video include the following steps:

1. Before watching the course video, a user must click a few points with the web camera operating to train up the model, known as calibration step
2. Then the user start watching the course video, while the prediction model works behind the scene and recorded his gazed position at every frame
3. When the user finish watching, gazed positions will be uploaded to server for storage and analysis

A flow chart illustrates the steps is shown below:

![Flow Chart](image)

*Figure 11: Primary Flow Chart*
4.4 Conceptual Class Diagram
The main conceptual classes and their entity relationships are shown in the following diagram, where\textit{GazesRecord} is the centre of the entire system.

![Conceptual Class Diagram](image)

4.5 System Components Diagram
As displayed in the following diagram, different system components were designed to support the preparation, retrieval, analysis and persistent storage of gazes records. \textit{Calibrator} is responsible to guide the user’s interaction to properly train our \textit{RidgeReg} (The regression model), \textit{Logger} stores runtime data and save it to persistent \textit{Storage}, for the use of \textit{HeatmapPlayer} and \textit{StatisticsModule}. 

![System Components Diagram](image)
4.6 Sequence Diagram
The main sequence diagram illustrates the process happened when a user watch a course video.
Tracking Learners’ Gazing Dynamics

Figure 14: Main Sequence Diagram
CHAPTER 5. Implementation

In this project, we used Node.js as our backend server and MongoDB as the supporting storage component. We coded Javascript for all client-side logic to be operated in a modern web browser. Below is an overview of the systems components.

5.1 Implementing the gaze-tracking solution

I worked out the implementation of the gaze-tracking solution based on a recent work Webgazer [16]. Webgazer is the first web-browser based eye tracking software developed by making use of only consumer grade web camera. It is a JavaScript based program that keeps calibrating and predicting continuously while a user is browsing a web page. The ridge regression model for gaze prediction was included in this package.

5.1.1 Problem of ClmGaze

However, its utility for online applications is still limited due to its high failure rate in the recognition of facial landmarks using ClmGaze, which was built upon the Clmtrackr library, and hence giving incorrect predictions often when we tried to integrate it into MOOC-Gaze. As shown in next chapter, ClmGaze cannot performed well in various environment on various type of faces.

5.1.2 Improvement: Customize jsFeatGaze

As an attempt to improve the existing problems of Webgazer, I implemented the proposed faces bound and eyes image extraction method with the aid of another JavaScript-based computer vision library - jsfeat, to replace the original facial landmarks locator ClmGaze. As a result, I customized my own eye extractor jsFeatGaze. This has been a complicated process and was explained in the following paragraphs step by step.

The first step was to hack into the internal structure of Webgazer for a source code analysis.

5.1.2.1 Source Code Analysis of Webgazer

Webgazer has been designed to have replaceable eye image extractors. It has the following interface:

```javascript
/**
 * Adds a new tracker module so that it can be used by setTracker
 * @param {String} name - the new name of the tracker
 * @param {Function} constructor - the constructor of the current module
 * @return {webgazer} this
 */
webgazer.addTrackerModule = function(name, constructor) {
    // Omitted verbose details
};
```
Thus, what I had to do was to build a new component, namely TrackerModule according to its interface specifications and then incorporate it into `Webgazer`. The interface specification of the TrackerModule was induced from `ClmGaze`, which has mainly this function, that provided eye images for the prediction model:

![Figure 15: Eye Extractor Interface](image)

### 5.1.2.2 Building a New Eye Image Extractor: `jsFeatGaze`

I employed `jsfeat` library to build a new eye image extractor, namely `jsFeatGaze`. In the official demo page `jsfeat` showed how to use it to perform haar face detection with its pretrained model.

#### 5.1.2.2.1 Wrapping `jsfeat` as a standalone Object

The demo page of `jsfeat` demonstrates how to use it on a plain web page together with web cam stream. However, `Webgazer` has its own video stream component already. Also, the demo page’s code contains lots of global variables where video parameters and image processing variables are tightly coupled, making it cannot be directly taken and plugged into `Webgazer`, but must be carefully crafted.

![Figure 16: jsfeat face detection demo: video-canvas logic, page statistics tool and detection logic crowded in the same namespace with shared variables](image)

I took the following steps to create an eye image extractor that can work perfectly with `Webgazer`:

```javascript
/**
 * Isolates the two patches that correspond to the user's eyes
 * @param {Canvas} imageCanvas - canvas corresponding to the webcam stream
 * @param {Number} width - of imageCanvas
 * @param {Number} height - of imageCanvas
 * @return {Object} the two eye-patches, first left, then right eye
 */
ClmGaze.prototype.getEyePatches = function(imageCanvas, width, height) {
    ...
}
```
1. Clone the demo page and change the input source of jsfeat from video directly to a flexible interface that can receive specified single-frame image
2. Split the variables that is shared between the video, page statistics tool and jsfeat face extraction functions
3. Remove the video component along with its variables
4. Test iteratively to ensure the removal does not affect jsfeat face extraction
5. After the video component and related variables was removed, wrap up the remaining part as a standalone object that aligns with the interfaces as the same as defined in the extractor based on Clmtrackr
6. Add in own-proposed eye locating method, as decribed in previous chatpers.

```javascript
var eyes = Detector(imageCanvas).findEyes(imageCanvas);
```

Figure 17: one line detector for flexible use after improving jsfeat demo code

Then I put it into Webgazer and ran the program to tried out the whole prediction.

5.1.2.2 Issues – dependency conflicts and asynchronous loading

However, the console report wired error which did not exist when in previous testing. After a series of tuning and debugging, I found out that there existed a conflict between ClmGaze and jsFeatGaze which made these two extractor could not exist in the same compiled Webgazer, which is the problem causing failures in the integration. This conflict was due to that ClmGaze used jsfeat’s earlier version as part of its dependency to assist in its own facial locating process. Since it shares the same namespace jsfeat with the one I am currently using, and the later one overrides the former one. This led to an inconvenience. As Webgazer designed, different extractors were supposed to co-exists in one package so that it may be used interchangeably or simultaneously. However, with this dependency conflict, it means that I have to compile two different version of a complete Webgazer to do comparison and tests separately.

To flexibly switch between two versions in run time, I cannot use conventional ways of loading scripts by writing `<script>` tags since it is statically written in the html page, while I need to dynamically load in scripts according to testing purposes. Dynamic script in browsers are loaded asynchronously, which means I cannot control the order they are executed. This was unacceptable since Webgazer must be loaded prior to other components, and there are also execution order dependency in other components.

I used the latest feature of modern browser – the support of Pormise, to achive the goal. A promise is an object which can be used to chain up asynchronous action in a designated order.
5.2 Building a Calibrator

Webgazer was not designed for MOOC. It was designed to be mainly used in web page browsing [16]. In its original design, when users browse the webpage, they move their cursors and click on different things, and the training model will be built by recording those cursor movements and clicks, assuming that they normally look at where they move their cursors [16].

However, in MOOC video lecture time, students do not have frequent cursor movements and clicks. Thus, an explicit calibration step was created before a user watch the video in order to train up the prediction model.

5.2.1 Support model training

The Calibrator receives two parameters, width W and height H, to split the whole screen evenly into W * H blue rectangles. These rectangles will be displayed one by one in random order for a user to click. When he clicks it, our prediction model gets one training sample. When all rectangles have been clicked, it will notify a “finish” signal for other components to know.

The calibrator was implemented by using HTML canvas element to draw blue rectangles onto the screen.
5.2.2 Support project experiments
In addition to its normal use case, the calibrator also has to support our project experiments. In experiments, we need to collect users’ click position independently from Webgazer, thus each time a user clicks a rectangle the Calibrator will emit an event along with user’s clicked coordinates. Any components, e.g. a logger, can listen to this event and receive these positions information.

5.3 Building VideoPlayer
VideoPlayer is a specific player used in MOOC-Gaze. The major features of this player are that it can maximumly scale the video automatically. Due to the limited accuracy of Webgazer, MOOC-Gaze cannot support a very small window size of video because it has prediction errors. Thus, every video loaded in will be scaled to fit in the screen maximumly, by either its width or height. Videos with different ratio needs to be scaled differently, the player will ensure maximum of area when scaling the video.

![Video scaling](image)

*Figure 19: Scaling videos to take maximum area*

5.4 Building HeatmapPlayer
HeatmapPlayer is a player which render heatmap overlaying onto the video. It has to monitor the play process of VideoPlayer and change its rendering according to gazed position data corresponding to the video frame that are being played. The rendering function depends on Heatmap.js, a third-party heat map renderer.
5.5 Building a Central Event Bus

Underlying the system, different components work independently and have their own states. The sequence diagram conceptually illustrated how they cooperate with each other. However, in the implementation they don’t directly talk to each other. When they have to work together, they communicate through a customized central event bus. I simplify the browse DOM event mechanism into two simple methods:

```
EventBus

+simpleBroadcast(name, data)
+onSimpleBroadcast(name, data)
```

```
var simpleBroadcast = function(name, data){}
var onSimpleBroadcast = function(name, callback){}
```

The event bus is essentially two method: one for broadcasting data, other for monitoring a specific channel of broadcast, and a callback action of when that event happens. Below shows a few important uses of the event bus.
The benefit using an event bus instead of direct invoke between components is to maintain a centralized communication platform for ease of management, also to keep a loose-coupled structure. A component should only have minimum knowledge about other components which does things beyond its own scope to support flexibility and maintainability. Taking sample usage 1 as instance, assumed that if we have another object that has to synchronize with the video frame playing, we just have to add one more listener with the new object without modifying the existing code.

Other implementation details are covered in vol. 2 of this report.
CHAPTER 6. System Screenshots

Figure 23: Publish course form - filled by course providers

Figure 24: Landing page - list of courses
Figure 25: Calibration sample: user has to click the blue rectangle

Figure 26: Calibration sample: user has to click the blue rectangle
Figure 27: Calibration sample: user has to click the blue rectangle

Figure 28: Calibration sample: user has to click the blue rectangle
Swapping Elements:

\[
\begin{array}{c}
\text{x} & \text{y} \\
3 & 3 \\
\end{array}
\]

Problem: Since \( y = x \) overwrites the value of \( y \), \( x = y \) has no effect. The value stored in \( y \), which is seven, is lost.

\[
\begin{array}{c}
\text{y} = \text{x} \\
\text{x} = \text{y} \\
\end{array}
\]

Figure 29: User watching a course video

Swapping Elements:

\[
\begin{array}{c}
\text{x} & \text{y} \\
3 & 7 \\
\end{array}
\]

Figure 30: Replay - Sharpen color represents more gazes at the same frame
Figure 31: Replay - Sharpen color represents more gazes at the same frame
CHAPTER 7. Experiments

In order to verify the outcome of replacing the old eye image extractor with the new one. A series of experiments were conducted, as described in the following paragraphs. The hardware and software set up used in experiments are listed below:

7.1 Testing of The Eyes Extraction

In this test, I chose a public dataset [19] and let the two eye image extractors identify the eyes of the frontal faces of the dataset. This dataset contains a total of 450 frontal images in environments with different spaces and illumination. This dataset is reasonable to be test samples since most of these images are close to this project’s actual use cases, compared to some other datasets which only include only already-cropped faces or faces in single-coloured background.

I manually labelled the iris of each face in the dataset and compared the eye region detected by two image extractors. As long as both eye region detected contained both irises, the detection would be considered success.

Encouragingly, jsFeatGaze successfully detected the eye regions of 426 out of 450 pictures, while Clmtrackr only got a total of 27 successful times of detection.

<table>
<thead>
<tr>
<th>jsFeatGaze (this study)</th>
<th>ClmGaze [16]</th>
</tr>
</thead>
<tbody>
<tr>
<td>95%</td>
<td>6%</td>
</tr>
</tbody>
</table>

Table 1: Comparison of the success rate of eyes extraction by jsFeatGaze and ClmGaze.

jsFeatGaze performed stably across environment with different extent of illuminations. This suggest that in it has a strong potential to work ideally in real-life use cases.

ClmGaze succeeded it detection in multiple images which belong to only two Asian people. It was doubted that ClmGaze has a strong bias in selecting faces, and would only worked well in the limited types of faces it favoured.
Figure 32: Eye extraction test results of 450 human face images by jsFeatGaze. Cases with successful eyes recognition are indicated with a blue rectangle at the top left corner of the image.
Figure 33: continued
Figure 34: continued
Figure 35: Eye extraction test results of 450 human face images by ClmGaze. Cases with successful eyes recognition are indicated with a blue rectangle at the top left corner of the image.
Figure 36: continued
Figure 37: continued
7.2 Testing of the Training Data Set Size on Gaze Prediction

To test the performance of the new solution, I did a preliminary experiment in the following way:

Experiment Setup:
- Computer Model: Lenovo T460P
- Camera Model: Built-in web camera of the computer 1280×720 (pixel) resolution
- Operating System: Windows 10 64-bit
- Browser: Google Chrome - Version 57
- Browser window resolution: 1707 * 893 pixels (32.2cm * 16.84cm)
- Distance between test user and screen: 60-70cm
- Number of test user: 1

The window was evenly divided into 50 pieces of blue rectangles, which the test user had to click one by one when each of them appeared on the screen in random order. When a user click the rectangle, his frontal image will be captured by the camera, as well as the coordinates of the position he clicked. Research showed that when a user click the screen he will naturally gaze at that clicked position, thus this position can be trusted as gaze target [16]. Such 50-point calibration will be carried out for 6 times to make up of 300 calibration points and images, as 6 calibration bags. Below shows a sample of user’s clicked positions distributed over the screen recorded when he clicked all blue rectangles.

![Test user image sample](image-url)
7.2.1 Large size of training points: 50 - 250

Later, the last bag was used as the test data set, while the remaining bags were used as training data set. Different number of points and its corresponding images were selected randomly to train the model and then tested against the test data set, ranging from 20 – 250, as demonstrated below. The error, measured in pixel distance, was calculated using the predicted coordinates and the clicked position (assumed true gaze coordinates) with the following formula:

\[ error_i = \sqrt{(x_{i1} - x_{i2})^2 + (y_{i1} - y_{i2})^2} \]

where \((x_{i1}, y_{i1})\) and \((x_{i2}, y_{i2})\) are the coordinates of predicted position and clicked position, respectively. In the case where eyes extraction fails and prompt null in the system, the error will directly be set to 2500 for ease of computation. The average error and standard deviation of the 50 errors were calculated as:

\[ error_{\text{average}} = \frac{1}{50} \sum_{i=0}^{50} error_i \]

\[ error_{\text{standardDeviation}} = \sqrt{\frac{1}{50} \sum_{i=0}^{50} (error_i - error_{\text{average}})^2} \]

Table 2 shows the averaged training errors of experiments and their standard deviation using different number training points. Our method jsFeatGaze learns correctly all training points given inputs of 20 to 100 training data. Interestingly, above 100 training points, the training error is drastically increased. In contrast, regardless of the number of training points, ClmGaze gives errors in the range of 500 to 1000 pixel distances with high variability.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.80</td>
<td>0.55</td>
<td>590.46</td>
<td>659.05</td>
</tr>
<tr>
<td>40</td>
<td>0.99</td>
<td>0.46</td>
<td>559.89</td>
<td>296.34</td>
</tr>
<tr>
<td>50</td>
<td>0.90</td>
<td>0.41</td>
<td>739.05</td>
<td>425.08</td>
</tr>
<tr>
<td>60</td>
<td>0.87</td>
<td>0.53</td>
<td>973.52</td>
<td>502.16</td>
</tr>
<tr>
<td>80</td>
<td>0.77</td>
<td>0.64</td>
<td>664.98</td>
<td>402.37</td>
</tr>
<tr>
<td>100</td>
<td>0.87</td>
<td>0.49</td>
<td>1158.07</td>
<td>747.46</td>
</tr>
<tr>
<td>150</td>
<td>64.40</td>
<td>36.62</td>
<td>1098.04</td>
<td>723.69</td>
</tr>
<tr>
<td>200</td>
<td>84.96</td>
<td>53.49</td>
<td>733.12</td>
<td>572.74</td>
</tr>
<tr>
<td>250</td>
<td>99.11</td>
<td>61.88</td>
<td>620.88</td>
<td>427.05</td>
</tr>
</tbody>
</table>

*Table 2: Comparison of training errors between jsFeatGaze and ClmGaze for large training set size.*

*Figure 40: Line plot of training errors of jsFeatGaze and ClmGaze for large training set size.*
After the calibration phase of certain number of training points, we tested the accuracy of gaze prediction with a set of 50 unseen points. Table 3 shows the testing errors of jsFeatGaze and ClmGaze. Our method jsFeatGaze has a consistently lower test errors compared to ClmGaze, yielding the average pixel distance error in the range of 200 to 400. ClmGaze has 2-3 times higher in distance error with high variation in predictions.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>299.93</td>
<td>188.37</td>
<td>784.26</td>
<td>709.37</td>
</tr>
<tr>
<td>40</td>
<td>277.61</td>
<td>162.90</td>
<td>568.83</td>
<td>440.64</td>
</tr>
<tr>
<td>50</td>
<td>226.80</td>
<td>145.65</td>
<td>451.06</td>
<td>288.43</td>
</tr>
<tr>
<td>60</td>
<td>316.93</td>
<td>210.73</td>
<td>504.59</td>
<td>329.66</td>
</tr>
<tr>
<td>80</td>
<td>306.82</td>
<td>212.05</td>
<td>677.47</td>
<td>389.90</td>
</tr>
<tr>
<td>100</td>
<td>396.69</td>
<td>220.16</td>
<td>1079.67</td>
<td>583.33</td>
</tr>
<tr>
<td>150</td>
<td>358.91</td>
<td>223.14</td>
<td>917.60</td>
<td>582.31</td>
</tr>
<tr>
<td>200</td>
<td>228.47</td>
<td>128.93</td>
<td>729.11</td>
<td>400.18</td>
</tr>
<tr>
<td>250</td>
<td>198.03</td>
<td>105.57</td>
<td>713.76</td>
<td>532.88</td>
</tr>
</tbody>
</table>

measured in pixels

Table 3: Comparison of testing errors between jsFeatGaze and ClmGaze for large training set size.
7.2.2 Small size of training points: 2 – 20

Since the less the calibration points, the better the user experience we can have. We conduct an experiment to see the change of error in smaller size of training points, hoping to see a user-friendly size of training points with acceptable error mean.

Table 4 shows the averaged errors of experiments and their standard deviation using different number of training points. Our method jsFeatGaze fits all training points given inputs from 2 to 20 training data. ClmGaze gives errors ranging from 200 to 1000 pixel distances with high variability again. The 2500 errors in the table indicates a failure of
ClmGaze attempts to identify a face for ease of calculation, in which case ClmGaze caused the prediction model to terminate.

<table>
<thead>
<tr>
<th>Number of Training Points</th>
<th>jsFeatGaze</th>
<th>ClmGaze</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>6</td>
<td>0.83</td>
<td>0.37</td>
</tr>
<tr>
<td>8</td>
<td>0.85</td>
<td>0.52</td>
</tr>
<tr>
<td>10</td>
<td>0.82</td>
<td>0.57</td>
</tr>
<tr>
<td>12</td>
<td>0.80</td>
<td>0.59</td>
</tr>
<tr>
<td>14</td>
<td>0.93</td>
<td>0.52</td>
</tr>
<tr>
<td>16</td>
<td>0.79</td>
<td>0.56</td>
</tr>
<tr>
<td>18</td>
<td>0.86</td>
<td>0.56</td>
</tr>
<tr>
<td>20</td>
<td>0.79</td>
<td>0.61</td>
</tr>
</tbody>
</table>

Table 4: Comparison of training errors between jsFeatGaze and ClmGaze for small training set size.

![Line plot of training errors of jsFeatGaze and ClmGaze for small training set size.](image)

Figure 44: Line plot of training errors of jsFeatGaze and ClmGaze for small training set size.
After the calibration phase of certain number of training points, we tested the accuracy of gaze prediction with a set of 50 unseen points. Table 5 shows the testing errors of jsFeatGaze and ClmGaze. Our method jsFeatGaze still has a consistently lower test errors compared to ClmGaze, yielding the average pixel distance error in the range of 200 to 500. jsFeatGaze error has an obvious trend to shrink as data size increases, while ClmGaze’s errors stably stays in the bigger range of 500-700 in such small number of dataset.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>477.12</td>
<td>251.47</td>
<td>2500.00</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>453.17</td>
<td>228.84</td>
<td>664.67</td>
<td>549.27</td>
</tr>
<tr>
<td>6</td>
<td>436.39</td>
<td>234.08</td>
<td>586.12</td>
<td>553.16</td>
</tr>
<tr>
<td>8</td>
<td>406.00</td>
<td>227.42</td>
<td>683.38</td>
<td>557.70</td>
</tr>
<tr>
<td>10</td>
<td>323.33</td>
<td>179.66</td>
<td>573.78</td>
<td>559.78</td>
</tr>
<tr>
<td>12</td>
<td>381.57</td>
<td>213.46</td>
<td>631.83</td>
<td>571.55</td>
</tr>
<tr>
<td>14</td>
<td>363.92</td>
<td>195.25</td>
<td>631.33</td>
<td>558.34</td>
</tr>
<tr>
<td>16</td>
<td>280.07</td>
<td>138.29</td>
<td>618.25</td>
<td>571.83</td>
</tr>
<tr>
<td>18</td>
<td>260.85</td>
<td>132.87</td>
<td>579.45</td>
<td>563.02</td>
</tr>
<tr>
<td>20</td>
<td>249.45</td>
<td>124.56</td>
<td>638.37</td>
<td>545.41</td>
</tr>
</tbody>
</table>

Table 5: Comparison of testing errors between jsFeatGaze and ClmGaze for small training set size.

Error over 1000 is truncated

measured in pixels
7.3 Discussion
The experiments showed that our improved Webgazer with jsFeatGaze outperformed the old solution (Webgazer with ClmGaze) in all cases. jsFeatGaze obtained its best
performance when the number of calibration points reached the maximum, which is an error of 198.03 ± 105.57 pixels, which is approximately 4 ± 2 cm error in average.

At point 20 we gained an average error of 248.45 ± 124.56 pixels, which is approximately 4.7 ± 2.35 cm physically.

At point 50 it showed an average error of 226.80 ± 145.65 pixels, which is approximately 4.28 ± 2.75 cm physically.

Although 250 points reached the best performance, it was impractical to ask a user to calibrate for 250 points before watching a video. The experiments revealed that for smaller size of training points 20 and 50, the average error was only slightly worse than the best performance, which has a larger distance not more than 1 cm. To make a concession between user experience and model accuracy, we eventually chose 20-point calibration for the system. In practice, it means that the system can be used to tracked users’ gazes on relatively large objects but not small items like a line of text.

As just a preliminary test due to limited time, it is not convincing enough to draw a solid conclusion. More experiments have to be conducted and documented in the future. Illumination of the environment, different type of faces, changes of head poses, faces with different type of glasses, etc. should be evaluated in terms of its impact on our solution’s performance.
CHAPTER 8. Ethics and Professionalism

8.1 Privacy of Data

In the current era of big data, any personal information may be used improperly and resulted in unexpected personal identity leakage. For a camera-based application like MOOC-Gaze, the first sensitive information is users’ images. We believe the best way to protect them is not to collect them.

We have carefully considered the criticalness of image information, and therefore, chose to develop a fully client-based gaze tracking solution. In MOOC-Gaze, all user images will never be transferred over network, and our server will never have a chance to touch these image data. A user image is processed only in the browser locally, and the prediction model is trained also locally.

What will be transferred to server is the gazes record, which contain information of video frames and users’ gazed positions. These are also privacy related issues, with which a user’s in-class behaviour may be investigated. Therefore, the system must ask for users’ consents in advance.

8.2 Credit to third-party code

As mentioned above, this project based its work upon Webgazer [16] package and jsfeat computer vision library which are under GNU General Public License(GPL) version 3 and MIT license respectively. We have also employed heatmap.js [20], also under MIT license, as to render heatmaps. Given the nature of GNU GPL license of Webgazer, this solution will also be made under GPL with a clear tracking of our modifications upon the original work and source code left opened.

8.3 Miscellaneous

The whole solution is not production-ready. Users must customize their own security control and data encryption when employing this solution so that no users’ gazes record will be leaked. Although the image capturing and processing are carried out locally in client’s side, as a web-based application it may still be under the attack of Cross-site scripting (XSS) once it is exposed in the Internet in unencrypted channel, e.g. HTTP. Enforcing HTTPS may be a good start point in the protection of users’ images to eliminate the threat of XSS.
CHAPTER 9. Conclusion

Gaze tracking is a long-existed topic in Human Computer Interaction. This method is often used in research fields with the aid of professional devices, for instance, customized cameras. However, gaze tracking with only a normal web-camera has not been adopted widely, and its first purely web-based solution was just published last year, which was Webgazer [16]. While Webgazer was designed in normal web browsing scenes, we pioneered this technique in education, that is, attempted to adapt gaze tracking into MOOC-platform based class. In the process of integration, we optimized the performance of Webgazer by second development, where we replace its original eye-image locator with our customized one, based on the work of another computer vision library, jsfeat.

Up to now, this solution can only identify relatively big objects in the screen (4-6cm, assuming a 1707*893 resolution according to test machine) in terms of accuracy. If a course video contains many small pieces of texts or widgets this solution may not be suitable. Different browsers have a different default scaling factor dependent on a screen’s resolution, this solution’s adaptability across different devices is yet to be tested.

Future directions of this project can include 1) improving the prediction model, for example, attempts to train a non-linear model. However, the performance of browser’s computability must be taken into considered since non-linear model can consume much more computing resources. 2) Using supervised-feature as input together with eye images, e.g. the distance between iris and eye border, the openness of eyelids [3], etc.

We developed this solution, hoping to provide a new way of MOOC-class feedback for course providers to evaluate the quality of course videos as well as students’ behaviour, as a compensation to the loss of face-to-face eye contacts in traditional classrooms. This way of feedback also has advantages over conventional web-class feedback channel like web forums which require MOOC student to do after-class reflection. With eye-tracking users’ gazed areas can be recorded without intruding users’ study, and the feedback will be just ready as they finish watching the video, by which course providers can know more about their students’ in-class behaviours even if they never submit a sentence in a forum. As an explorative attempt to introducing a new way of feedback for MOOC education, this project has a great potential to be further developed to contribute to online learning industry.
REFERENCES


