UNIVERSITY OF CAMBRIDGE
Department of Engineering

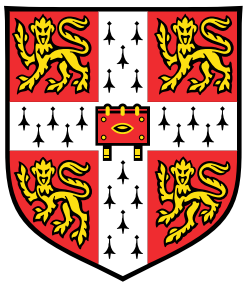
Webcam-based Eye Tracking for Psychophysics Experiments

Author Name:Youjing Yu
Supervisor:Dr Guillaume Hennequin
Date: 30/05/2023

I hereby declare that, except where specifically indicated, the work submitted herin is my own original work.

Signed _____ *date* _____30/05/2023_____

Webcam-based Eye Tracking for Psychophysics Experiments



Youjing Yu

Supervisor: Guillaume Hennequin

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Engineering

Trinity College May 2023

Acknowledgements

I am sincerely grateful to my supervisor, Dr Guillaume Hennequin for his support and advisory on this research project. Without his invaluable guidance, this project would not have been possible. I would also like to specifically acknowledge that the code for the WebGazer workflow is largely based on the original work of WebGazer [20] and the code for the LPD and blind spot calibration in Chapter 3 is based on the work of Li et al. [19].

Abstract

Visual psychophysics experiments study the relationship between the physical world and human behaviour and have found widespread applications in neuroscience and behavioural science research. Among a plethora of visual psychophysics techniques, one of the most popular is eye tracking, which determines the location of the user's gaze on a computer screen during browser sessions. In recent years, research on eye trackers has seen a shift from infrared-based methods which require users to wear a specialised headset to modern, computational methods which estimate the gaze location on the screen directly from webcam images. There are multitudinous advantages of webcam-based eye trackers over infrared eye trackers: they are cheaper, more convenient and impose minimum user intrusion. However, the accuracy of gaze estimation obtained from webcam-based eye trackers is also significantly worse compared to their infrared counterparts [6]. In addition, the underlying algorithms for webcam-based eye trackers may also be computationally costly and thus cannot be implemented in real-time. Hence, in our project, we aim to design a real-time webcam-based eye tracker which is able to achieve high accuracy, while maintaining low levels of user intrusion and is thus convenient for implementation.

We start by carrying out a feasibility study on WebGazer [20], one of the most popular webcam-based eye tracking software in the research community, on a 4 by 4 navigation task that is used as a benchmark. Ideally, the eye tracker should be accurate enough to predict in which square the user's gaze has fallen. From the feasibility study we conclude that the current implementation of WebGazer is not adequate for this benchmark task. Recognising its deficiency, we propose to improve its performance in two aspects: **calibration** and **algorithm**. Instead of directly mapping from the webcam images of the eye to the gaze locations on the screen (as in WebGazer), we propose to (1) gauge the relative position of the user with respect to the screen and (2) map from the webcam images of the eye to the eye rotation angles in the horizontal and vertical directions. Knowledge of both the relative position and rotation angles gives us the gaze location on the screen through simple geometric calculations.

In the calibration stage, we implement a simple card slider calibration and a blind spot calibration to gauge the relative position of the user, which is inspired by the virtual chin-rest idea proposed by Li et al. [19]. In the algorithm stage, instead of the linear regression model originally implemented in WebGazer, we propose the use of a Gaussian Process (GP) regressor to map from the eye features (obtained from the webcam images) to the rotation angles. We collect eye tracking data on three users and train the GP regressor offline with the squared exponential (SE) kernel, the rational quadratic (RQ) kernel and a novel kernel, the separable smoothing (SS) kernel proposed by ourselves. All three kernels achieve similar performance, and we report

a mean absolute error of 1.43° for the SE kernel for the combined data set, which is a 22.9% improvement from the linear regressor. We also perform cross-validation between data sets collected from different users and conclude that optimal parameters obtained from the combined data set (from all users) generalise well to individuals.

Finally, we implement our improved calibration protocol and inference algorithm in the JavaScript package. We use the pre-determined optimised parameters obtained from the combined data set and directly perform inference, while conditioning on the data of the current user collected during a points clicking calibration exercise at the start of the user browser session. The SE kernel achieves the overall best performance. Compared to the original WebGazer implementation, we report a reduction in error in x-coordinate from 153 pixels to 91 pixels (40.5%) and in y-coordinate from 130 pixels to 43 pixels (66.9%), demonstrating the superiority of our proposed algorithm.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Behaviour neuroscience applies the principles of biology to the study of physiological, genetic, and developmental mechanisms of behaviour in humans or animals [30]. Research in behavioural neuroscience aims to understand the conceptual basis of human or animal behaviour by developing associative theories to provide explanations as to how low-level neural processes translate to key phenomena such as learning, attention and memory. To understand and validate the behaviour of biological systems, many visual psychophysics experiments have been designed and carried out, and eye tracking of participants is one of the most widely used methods. Eye tracking is an experimental technique of recording eye motion and gaze location across time and tasks. The position of our gaze is influenced by cognitive processes beyond attention, such as perception, memory, language, and decision making [7]. Studies have demonstrated that it is generally true that the eyes reflect the mental processing of what we are looking at [22, 29]. Hence, eye tracking provides knowledge of one's homeostatic balance and gate-keep information that shape decisions [32].

However, eye detection and tracking remains challenging despite active research in the field for over 30 years, due to reasons such as the individuality of the eyes, different lighting conditions, location and occlusion [11]. Hansen and Ji [11] proposed that gaze detection techniques can be grouped into two categories, namely feature-based approaches and appearance-based approaches. Feature-based approaches first extract local features such as eye contours, corners, or reflection from the eye image, before building models on the features to predict the gaze location. In feature-based eye tracking, one of the most popular methods is infrared eye trackers, which can determine the location of gaze with a high degree of accuracy by measuring the position of the corneal reflection of an infrared light relative to the pupil [16, 27]. IR-based methods take advantage of the spectral properties of the eye under near infrared (NIR) illumination. When NIR light is shone onto the eyes, it is reflected off the different structures in the eye and creates different types of IR illuminated eye features. Features that may be of interest include the limbus, pupil and images reflected by the cornea [14]. For example, Brousseau et al. [4] designed an infrared eye-tracking system that uses an industrial prototype smartphone with integrated infrared illumination and a camera. Zhu et al. [35] made use of combined conventional object recognition and tracking method with Kalman filtering to improve the performance of an

infrared eye tracker. Hansen and Pece [12] proposed an active contour tracker combining particle filtering and Expectation-Maximisation (EM) algorithms using infrared light as well. However, infrared-based methods for eye tracking are both expensive and inconvenient. Users would often need to come to the lab and perform the tasks in a specific environment setting with specialised equipment, which is highly constrained to specialised labs [14]. Moreover, sunlight and glasses can seriously disturb the reflective properties of IR light [12], making IR-based predictions less reliable.

Despite the popularity of IR light techniques, another category of cheaper and more convenient methods that entirely forgo the use of IR light is gaining attention. These are termed appearance-based approaches. Instead of explicitly extracting features, appearance-based methods use image contents from web cameras as input and map them directly onto the screen coordinates, in the hope that the implemented algorithms are able to implicitly extract the relevant features. This is made possible by the advancement in computational powers and resources, and the increased support for webcams in laptops and HTML5 websites. Appearance-based approaches forgo the need for camera calibration and are hence more convenient and desirable, thus they have been widely adopted in the research domain. For instance, Jung et al. [17] investigated users' engagement with interactive persona systems with webcam-based eye-tracking software. Rozsa [24] also examined how behaviour contingent praise affects visual engagement with an online video lecture with a webcam-based eye tracker. However, unsurprisingly, the prediction accuracy using webcam-based videos has been found to be much less than those obtained using specialised headware. In addition, software that rely on computationally expensive algorithms such as large neural networks are yet to achieve real-time eye tracking capabilities.

Building on these previous researches, our project aims to develop and implement a webcam-based eye tracker. The ideal eye tracker should satisfy three requirements: (1) The eye tracker should require minimal hardware and user intrusion to allow maximal flexibility, in particular it should be possible to crowd source the tasks on online platforms using the developed tracking software; (2) The eye tracker should achieve an accuracy level better than or at least comparable to that of existing eye tracking software. In our project we will use the performance of WebGazer [20] as a baseline; (3) The software should be computationally cheap enough to be implemented in real-time on consumer computers. In Chapter 2, we perform a review of WebGazer, an open-source JavaScript package that is one of the most popular eye trackers in the current research community. We design and carry out three accuracy tests to decide whether WebGazer is adequate for a benchmark navigation task and conclude that WebGazer is inadequate for the benchmark task. We proceed to explain its algorithmic structure and point out a serious limitation, which is the fact that information on the user's relative location with respect to the screen is not taken into consideration when estimating the gaze location. Building up on this observation, instead of directly mapping from the webcam images of the eye to the gaze locations on the screen (as in WebGazer), we propose to (1) measure the relative position of the user with respect to the screen and (2) map from the webcam images of the eye to eye rotation angles in the horizontal and vertical directions. Knowledge of both the relative position and rotation angles

allows us to calculate the gaze location on the screen through simple geometric relationships. In Chapter 3 we describe our proposed calibration algorithm where the user's relative position with respect to the screen is obtained by performing a simple card slider and blind spot calibration task. In Chapter 4 we explain the mapping from webcam images to eye rotation angles with a Gaussian Process (GP) regressor. We collect data on participants and train the GP regressor with the squared exponential (SE) kernel, the rational quadratic (RQ) kernel and a proposed separable smoothing (SS) kernel. A cross-validation experiment is also carried out to test the generalisation performance of the GP regressor. We finally decide to implement the SE kernel after comparing both the performance and computational load of the three kernels. In Chapter 5 we validate the performance of our improved implementation of WebGazer by comparing it with the original implementation. We report a reduction in error in x-coordinate from 153 pixels (px) to 91 px (40.5%), and in y-coordinate from 100 px to 43 px (66.9%), demonstrating the superiority of our proposed algorithm.

Chapter 2

Review of WebGazer

Due to its simplicity and ease of implementation, WebGazer is a widely used package in the research community for eye tracking [33, 21, 8]. In this section, we first give an overview of the algorithmic structure of WebGazer. Next, we describe the three accuracy tests carried out on WebGazer to decide whether it is adequate for the benchmark task (Figure 2.1) where the user navigates a 4 by 4 maze. Ideally, WebGazer should be accurate enough to determine in which square the user's gaze has fallen. Finally, we explain why WebGazer only achieves limited performance.

2.1 WebGazer Algorithm

We summarise the WebGazer algorithm in a flowchart shown in Figure 2.2. The video stream from the webcam, or the webcam pictures, is constantly fed into the MediaPipe Facemesh tracker. This tracker will in turn identify 468 key points on the human face in the video stream. The points identifying the inner arc of each eye are used to crop out the image of each eye and the image is re-scaled to a 6×10 image patch. The left and right eye images are concatenated to form a 6×20 image ($\mathbf{x}$ in Equation 2.1). This concatenated image is then greyscaled and histogram equalised to form the 120 dimensional (120 D) feature set ($\phi(\mathbf{x})$). When a user interaction is registered (for example clicking on a point on the web page during the calibration phase), the mouse location is recorded and this is deemed as the ground truth for the gaze position (g_x, g_y), with the assumption that the user's gaze position aligns with the mouse click location. The concatenated eye images and the click locations are then fed into the database to train a simple linear ridge regression model. Consider the ridge regression function for the x-coordinate prediction $f(\mathbf{x}) \to g_x$:

$$f(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{w} \tag{2.1}$$

where $\mathbf{w}$ is found by minimising

$$\sum_{x_i \in \mathbf{x}} ||g_{x_i} - f(\mathbf{x}_i)||_2^2 + \lambda ||\mathbf{w}||_2^2 \tag{2.2}$$

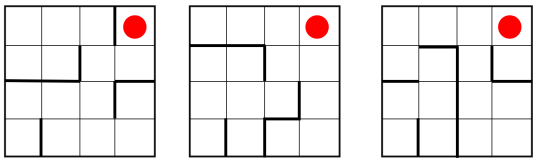


Fig. 2.1 Examples of the 4 by 4 maze benchmark task. The red dot represents the reward. Each small square is of dimension 100 px by 100 px, which is 23.8 mm by 23.8 mm on our setup.

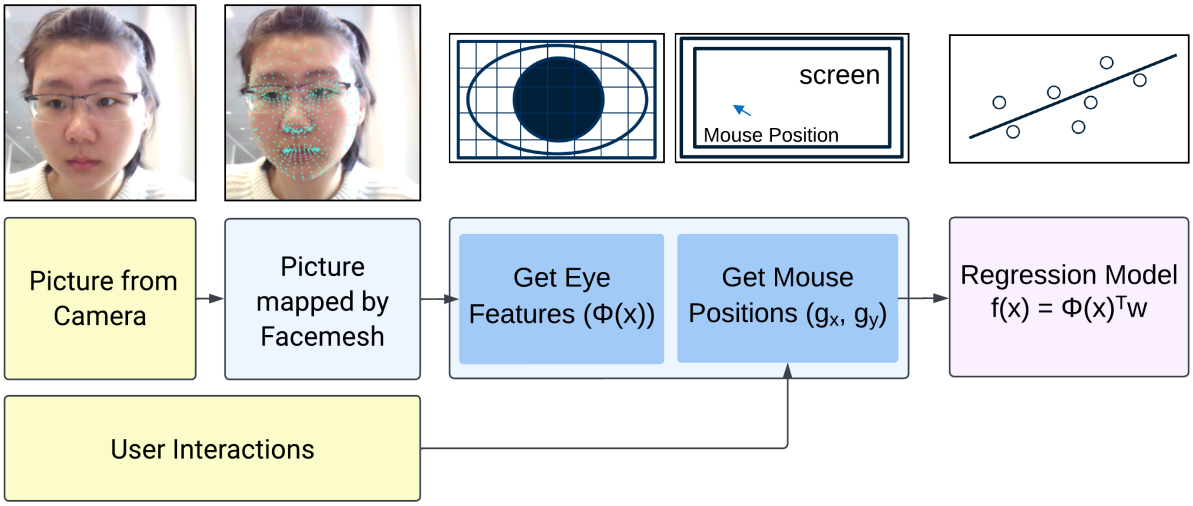


Fig. 2.2 Flowchart for the model training algorithm of WebGazer.

where λ acts as a regularisation term to penalise overfitting and is set to 10^{-5}. The new gaze position $f(\mathbf{x}_*)$ is then predicted as

$$f(\mathbf{x}_*) = \phi(\mathbf{x}_*)^T \mathbf{w}. \tag{2.3}$$

Note that a Kalman filter is implemented to smooth predictions and all following experiments are performed with the Kalman filter incorporated. Predictions of the current gaze location are made and painted as a red dot onto the screen every 50 millisecond (ms), or at a frequency of 20 Hz.

2.2 Accuracy Tests

A points clicking calibration exercise demonstration is provided in the WebGazer package released, as shown in Figure 2.3 (a). There are nine red points shown on the web page. Users are required to click each point five times until it turns from red to yellow while they are specifically instructed to stare at the point they are clicking. The ground truth data points are collected in this procedure and these are used to train the ridge regression model, as explained in Section §2.1. Using the same calibration procedure, we now design and implement one quantitative and two qualitative accuracy tests to gauge the performance of WebGazer, which will be explained below.

Note that the screen resolution is set to 1920×1080 on a 13.3-inch monitor (width 325 mm and height 325 mm) for the whole of this project. The typical viewing distance is 50 cm.

2.2.1 Quantitative Test

Our first test is carried out to quantify accuracy and error. After the calibration phase, a yellow dot will appear on the center of the screen. The user is then instructed to stare at the yellow dot on the screen for five seconds while the model predicts the gaze locations during the five-second window using the learned weights $\mathbf{w}$. These predictions are then used for performance evaluation. A demonstration is shown in Figure 2.3 (b). We now discuss the performance using two different metrics.

Error We define the error as the mean of the absolute distance between the predicted location and ground truth in the horizontal (ε_x) and vertical (ε_y) directions respectively. Formally, given the tolerance t, the ground-truth value of the gaze location, denoted as (g_x, g_y), and the prediction result, denoted as $(\hat{g}_x, \hat{g}_y)$, ε_x and ε_y are defined as

$$\varepsilon_x = \frac{1}{N}\sum_{i=1}^{N} |\hat{g}_{x_i} - g_{x_i}| \quad \text{and} \quad \varepsilon_y = \frac{1}{N}\sum_{i=1}^{N} |\hat{g}_{y_i} - g_{y_i}|. \tag{2.4}$$

where N is the sample size (total number of predictions). The author has repeated this experiment five times to ensure repeatability and the results are summarised in Table 2.1. We report an average value of ε_x to be 153 px and ε_y to be 130 px. In addition, WebGazer reports an average error of 4.17° in their online study, which translates to an error of 153 px on our setup with a typical viewing distance of 50 cm. This confirms that our feasibility test yields roughly the same error as reported by WebGazer hence the setup is working correctly. We also propose an additional accuracy metric that is more in line with our benchmark navigation task (Figure 2.1).

Accuracy We propose the accuracy metric where we deem the predicted gaze location to be accurate if the distance between the actual gaze position and the prediction is smaller than half the tolerance level in both the x and y coordinates of the screen. For our case we set the tolerance to be 100 px, which is the width of the square in the two accuracy games in Section §2.2.2 ($t = 100$). This measure is convenient because in the benchmark navigation task we wish to determine in which square the participant's gaze has fallen. If the error is larger than half the width of the square, we would not be able to predict in which square the gaze has fallen accurately. Mathematically, the accuracy metric α_t is defined as

$$\alpha_t = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{[g_{x_i}-t/2, g_{x_i}+t/2]}(\hat{g}_{x_i}) \cdot \mathbb{1}_{[g_{y_i}-t/2, g_{y_i}+t/2]}(\hat{g}_{y_i}), \tag{2.5}$$

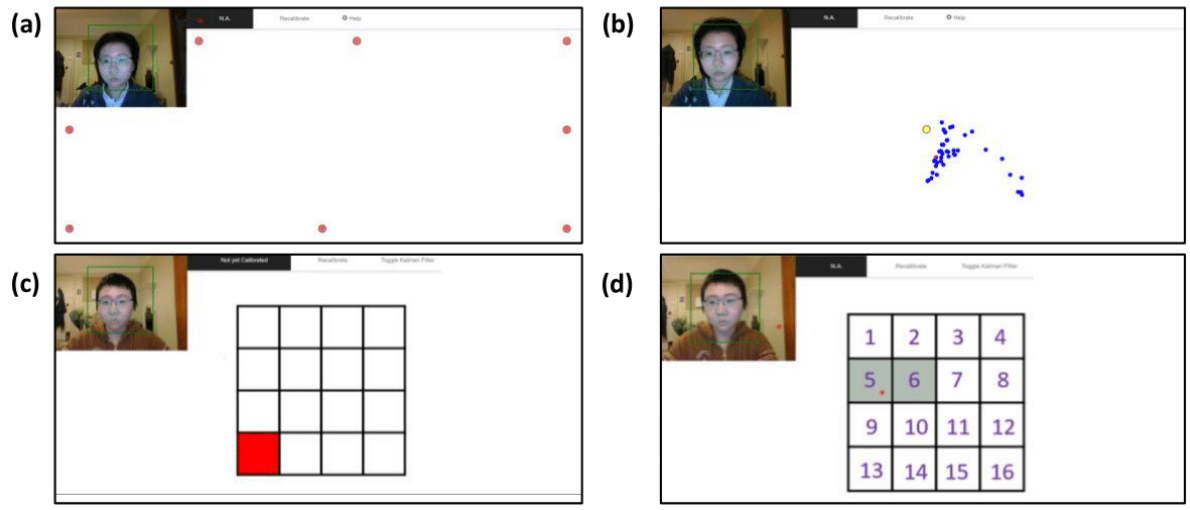


Fig. 2.3 (a) Calibration page for WebGazer implementation. The user clicks on each of the nine points on the edges of the screen five times until they turn yellow while staring at the point. (b) Accuracy test. The user is instructed to stare at the yellow dot while the algorithm predicts the gaze locations and paints them on the screen. The blue points represent the trace of predictions, the red dot represents the current prediction and the yellow dot is the ground truth gaze location. (c) Task page for game *Chase the Red*. (d) Task page for game *Trace the Shape*. Ideally squared numbered 1, 5, 9, 13, 14, 15, 16 should be painted grey-green. However, in this particular run, squares numbered 5 and 6 are painted grey-green, showing that the algorithm has wrongly determined the gaze to have fallen squares that the user did not look at.

Table 2.1 Calibration test results for the original WebGazer package. The definitions of metrics α_t, ε_x and ε_y and the experimental setup is explained in Section §2.2.1.

	$\alpha_{100}(\%)$	$\varepsilon_{\mathbf{x}}$ (px)	$\varepsilon_{\mathbf{y}}$ (px)
Trial 1	4	193	81
Trial 2	10	108	139
Trial 3	4	164	189
Trial 4	6	126	140
Trial 5	6	176	100
Average	**6**	**153**	**130**

and $\mathbb{1}$ is the indicator function satisfying

$$\mathbb{1}_{[a,b]}(x) = \begin{cases} 1, & \text{if } x \in [a,b] \\ 0, & \text{otherwise.} \end{cases} \tag{2.6}$$

It is clear that the performance is lacking: with an average accuracy of only 6% and ε_x of 153 px and ε_y of 130 px, WebGazer will not be able to accurately predict in which square the user's gaze has fallen.

2.2.2 Qualitative Tests

We also design two qualitative tests to give a visual impression of the accuracy level of WebGazer. For both games the dimension of the small square is 100 px by 100 px. A screenshot of the author performing the tests is shown in Figure 2.3 (c) and (d).

Chase the Red The first test is a game named *Chase the Red*, where a random square inside the maze will be painted red. As soon as the model predicts that the user's gaze falls inside the red square, a new red square will appear at another random location and the user is instructed to always look at the red square. When the author was playing this game as a feasibility study, it was noticed that it was very hard for the model to predict the correct square that the gaze has fallen in.

Trace the Shape The second test is a game named *Trace the Shape*, which asks the user to trace the shape of an L on the maze, starting from the top left corner and ending at the bottom right corner. The maze is initially unpainted. As soon as it detects the user's gaze falling inside a square, the predicted square will be painted a grey-green colour. In this game, we notice that the algorithm oftentimes predicts the gaze to be inside wrong squares and hence those wrongly predicted squares would be painted grey-green.

2.3 Explanation for Low Accuracy

In the previous section we demonstrated that the performance of WebGazer is lacking through the three accuracy tests. There are several possible reasons for the low accuracy. Firstly, the simple linear regression model cannot capture the inherent non-linear relationship between eye features and rotation angles. A more sophisticated model is called for to account for this non-linearity. This is analysed in more detail in Chapter 4. Secondly, and more importantly, there is missing information not captured in the WebGazer algorithm. The gaze location on the screen is dependent on two factors, **rotation** and **location**. Rotation refers to the angles of the eyeball in the horizontal and vertical planes and location refers to the relative location of the eye with respect to the screen. However, the only information that can be possibly inferred from the eye images is the eye rotation. For a more accurate prediction of gaze location, both the location and rotation of the eyes need to be considered. The methods for incorporating information on relative location are discussed in more detail in the next chapter.

Chapter 3

Improved Measurement and Calibration Protocol

In the previous chapter we recognised the inadequacy of the current WebGazer module and concluded that for a more accurate prediction information on the user's relative position with respect to the screen must be incorporated. However, knowledge about the user's physical setup, for instance the viewing distance, is not easy to obtain. Traditionally, experiments need to be conducted in the lab and participants would often be instructed to stay still to keep the same posture or use headsets, bite-bars or chin-rests [11]. This is so that information on the physical setup can be easily measured and kept track of. Ideally we would like to be able to track eye movements with minimal hardware (i.e. consumer computer camera) and impose minimal intrusiveness and obstruction, allowing for free head movements while maintaining high accuracy. Hence, we need to design methods that can infer the user's physical setup. In this chapter we will first give an overview of our proposed calibration protocol inspired from [19, 26], before explaining in detail how we measure and keep track of the relevant physical quantities. Finally, we carry out the accuracy tests described in Section §2.2 to gauge whether the proposed calibration protocol improves performance.

3.1 Calibration Protocol

In our implementation, we assume that the head of the user is completely parallel to the screen surface, which reduces the problem from three-dimensional to two-dimensional and simplifies the setup. To calculate the gaze coordinate on the screen, we need six quantities:

1. Viewing distance v_d, which is the perpendicular distance from the user to the screen (Figure 3.1 (a));

2. Horizontal eye distance x_d, which is the distance from the user's eye to the left edge of the screen (Figure 3.1 (a));

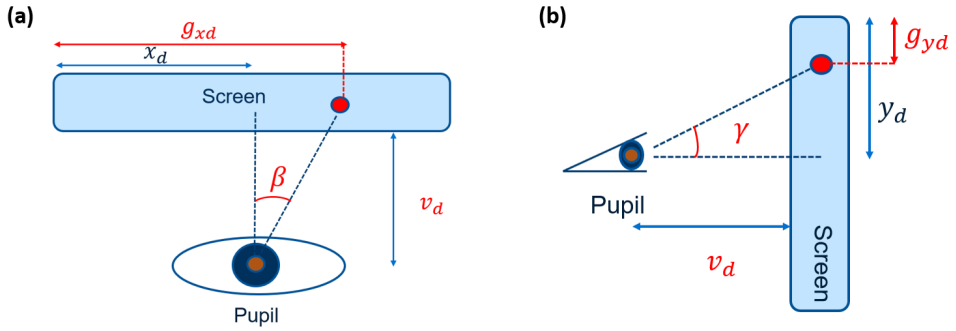


Fig. 3.1 (a) Trigonometric calculation of the horizontal (top view) and (b) vertical (side view) gaze position/rotation. Note that the red dot is the predicted gaze location. All symbols are defined in Equation 3.1.

3. Vertical eye distance y_d, which is the distance from the user's eye to the top edge of the screen (Figure 3.1 (b));
4. β, which is eye rotation angle in the horizontal direction (Figure 3.1 (a));
5. γ, which is eye rotation angle in the vertical direction (Figure 3.1 (b));
6. Logical Pixel Density (LPD), which is the length scale conversion factor. It is defined as the number of pixels each millimeter in real-life maps onto the screen;

The first three quantities, when combined, give us the relative position of the user with respect to the screen. The physical coordinate of the gaze (g_{xd}, g_{yd}) can then calculated as (Figure 3.1):

$$g_{xd} = x_d + d \times \tan(\beta), \quad g_{yd} = y_d - d \times \tan(\gamma). \tag{3.1}$$

Finally, we can convert from the physical coordinate to the pixel coordinate of the gaze location With LPD. The gaze location on screen (g_x, g_y) is calculated as:

$$g_x = \frac{g_{xd}}{\text{LPD}}, \quad g_y = \frac{g_{yd}}{\text{LPD}}. \tag{3.2}$$

The calibration procedure used to measure the LPD, v_d, x_d and y_d is shown in a flowchart in Figure 3.2. We now explain in detail how each of the quantities is obtained.

3.1.1 Length Scale Calibration

One of the main challenges in conducting psychophysical experiments online is that the resolution and size of the display used by each user are different. Hence the first step is to measure the LPD. We ask the users to place an Identity Card (ID) on the screen and scale the slider until the lengths match, which is shown in Figure 3.2. This is because the universality of card size (86 mm) provides a convenient length calibration standard. The LPD is then calculated as

$$\text{LPD} = \frac{\text{On-screen Distance (px)}}{\text{Physical Distance (mm)}} = \frac{\text{Card Image Width}}{\text{Actual Card Width}}. \tag{3.3}$$

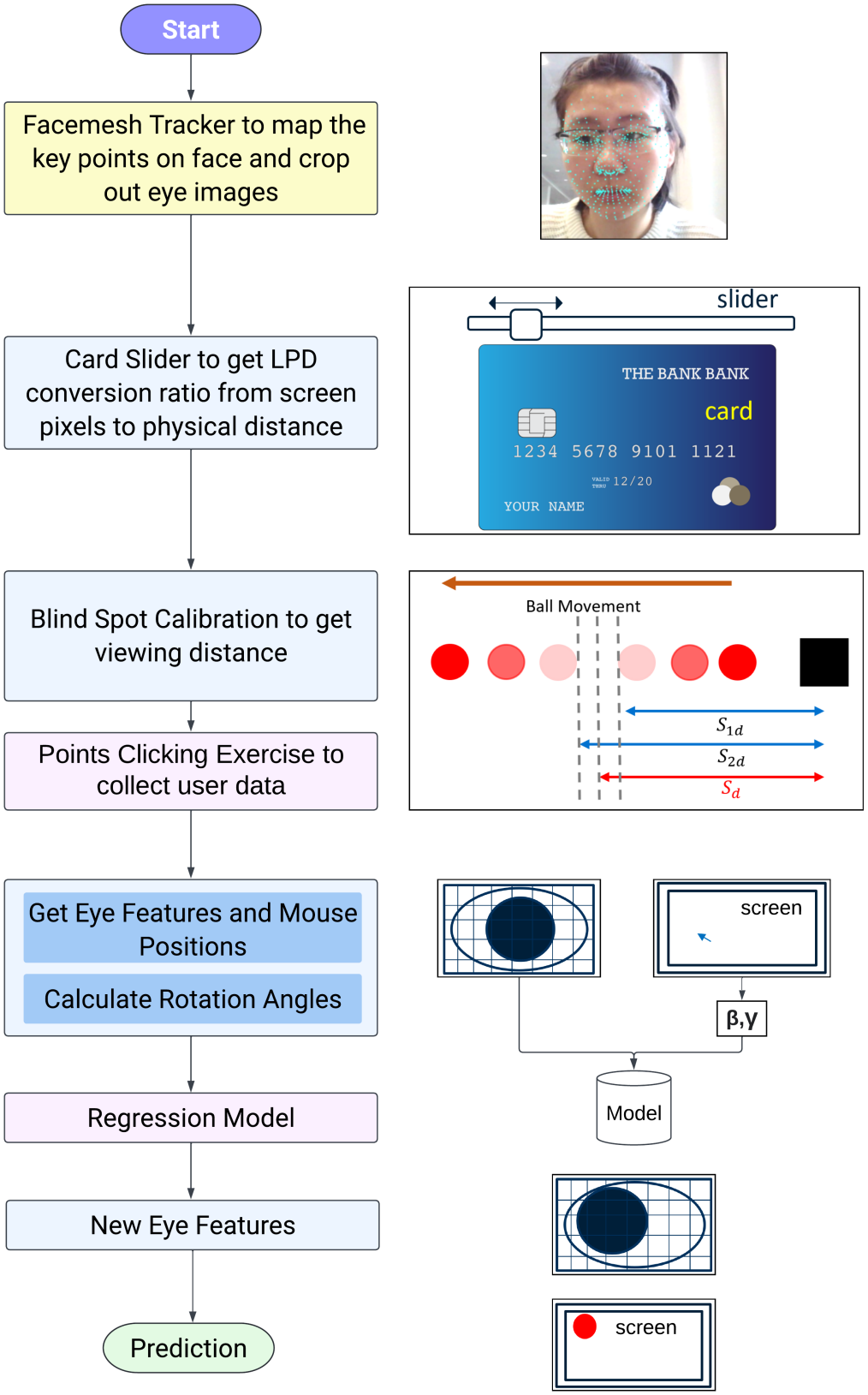


Fig. 3.2 Flowchart for the proposed algorithm flow.

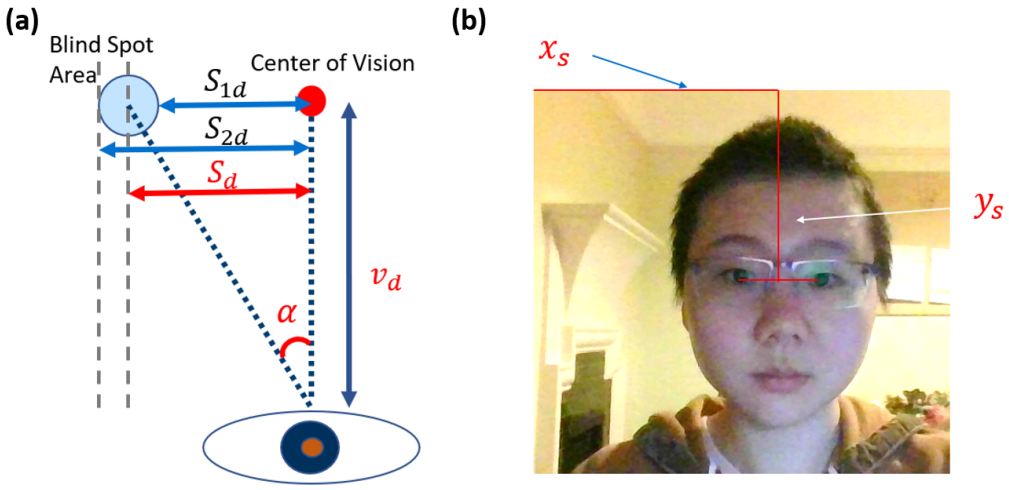


Fig. 3.3 (a) Trigonometric calculation of the viewing distance v_d in blind spot calibration. (b) Measurement of horizontal and vertical eye distances x_s and y_s from the webcam image. They are first measured in pixels and then converted to physical distances after division by LPD to get x_d and y_d.

This LPD value is then used in all following calculations to convert physical distances to/from pixel lengths.

3.1.2 Viewing Distance

The viewing distance is another factor that is difficult to measure and control in online experiments. Motivated by Li et al. [19], we measure the viewing distance by a simple blind spot calibration which uses the fact that the entry point of the optic nerve on the retina produces a blind spot where the human eye is insensitive to light, and the center of the blind spot is located at a relatively consistent angle of around 15.5° [1]. We now demonstrate the calculation of the viewing distance from the blind spot calibration.

On the web page there will appear a fixed black square and a moving red circle sweeping from right to left (Figure 3.2). Users are asked to fixate on the black square with the right eye closed (and left eye open) and respond (by pressing keys on the keyboard) as soon as they perceive that the red circle disappears (first response) and reappears (second response). The two events correspond to entering and leaving the blind spot region and we record S_{1s} and S_{2s} respectively. The trigonometric calculation of the viewing distance is shown in Figure 3.3 (a). Using Equation 3.3, we convert S_{1s} and S_{2s} from pixel lengths to physical distances as:

$$S_{1d} = \frac{S_{1s}}{\text{LPD}}, \quad S_{2d} = \frac{S_{2s}}{\text{LPD}}. \tag{3.4}$$

[1] Researches have reported the blind spot angle to be located at 15.5° ± 1.1[23], 15.48° ± 0.95 [25], 15.52° ± 0.57 [9]).

S_d is then calculated as the average of S_{1d} and S_{2d}:

$$S_d = \frac{(S_{1d} + S_{2d})}{2}. \tag{3.5}$$

Finally, we calculate the viewing distance as

$$v_d = S_d \times \tan(15.5^\circ). \tag{3.6}$$

We also ask the users to repeat the blind spot experiment five times and the average of the five measurements of S_d is used as the final viewing distance. In the original implementation by Li et al. [19] only S_{1s} (entering region) is used for the calculation of v_d. However, we decide to record both the entering (S_{1s}) and leaving (S_{2s}) points of the blind spot region since more research is done on the angle of the center of the blind spot region compared to the entering point [1, 5].

We now explain how we dynamically update the user's viewing distance after the initial viewing distance has been obtained from the blind spot calibration. If the user's viewing distance changes during the experiment, we leverage the fact that the width of the eye in the webcam image changes as well, assuming the physical setup does not change (i.e. the computer remains stationary on the table). We set the initial viewing distance as d_0 and record the initial width of the eye (both recorded at the time the blind spot calibration is completed). Hence we can recalculate the viewing distance as:

$$d_1 = d_0 \times \frac{\text{Initial Eye Width}}{\text{Current Eye Width}}. \tag{3.7}$$

The viewing distance is updated for each frame (at a frequency of 20 Hz for our experiments).

3.1.3 Eye Location

With LPD we can easily convert from the pixel distances x_s and y_s (Figure 3.3 (b)) to the physical distances x_d and y_d (Figure 3.1) [2]. Mathematically:

$$x_d = \frac{x_s}{\text{LPD}},\ y_d = \frac{y_s}{\text{LPD}}. \tag{3.8}$$

3.2 Accuracy Test and Results

The author now repeats the same accuracy tests as described in Section §2.2 with the added calibration steps described above. To obtain eye rotation angles β and γ, we use the same ridge regression model explained in Section §2.1. The input for the model remains unchanged (eye images $\mathbf{x}$). However, instead of mapping from $\mathbf{x}$ to the gaze location (g_x, g_y) ($f(\mathbf{x}) \rightarrow g_x, g_y$),

[2] Note that for all symbols defined, subscript d denotes *physical distances* and subscript s denotes *pixel lengths* on screen.

we now map from $\mathbf{x}$ to the rotation angles (β, γ) ($f(\mathbf{x}) \rightarrow \beta, \gamma$). The results are summarised in Table 3.1. Compared to the initial WebGazer implementation, the accuracy has improved from 6% to 8%, ε_x is reduced from 153 px to 137 px, and ε_y is reduced from 130 px to 58 px. The improvements in errors in the horizontal and vertical directions give us confidence in the validity of the improved calibration protocol. However, this level of accuracy is still inadequate for the navigation task that we set as our benchmark. To further improve the prediction accuracy, we propose the use of a Gaussian Process regressor in place of linear regression. This is discussed in more detail in the next chapter.

Table 3.1 Results for proposed calibration phase with ridge regression. The definitions of metrics α_t, ε_x and ε_y and the experimental setup is explained in Section §2.2.1. For comparison of results obtained with the original WebGazer implementation refer to Table 2.1.

	$\alpha_{\mathbf{100}}(\%)$	$\varepsilon_{\mathbf{x}}$ (px)	$\varepsilon_{\mathbf{y}}$ (px)
Trial 1	2	177	47
Trial 2	6	148	64
Trial 3	20	159	67
Trial 4	12	99	58
Trial 5	2	103	54
Average	**8**	**137**	**58**

Chapter 4

Improved Inference Protocol

In Section §3.2, we used the same ridge regression model as the original WebGazer to map from the eye features to the rotation angles and we have concluded that the performance is still lacking. Here we propose the use of Gaussian Processes (GPs) [31]. Compared to the linear regression model, GPs are able to model non-linear relationships and are thus ideal for our task. They are also easy to train and computationally tractable. We will first briefly describe what a Gaussian Process is, before explaining the structure of the three kernels for the GP regressor that we have experimented with. We will then compare and discuss the regression results. Finally, we perform cross-validation across subjects and discuss their results and implications on the implementation.

4.1 Gaussian Processes Definition

A Gaussian Process is an (infinite) collection of random variables $f(\mathbf{x})$, where f maps from an input space $\mathscr{X}$ to an output space $\mathscr{R}$, such that every finite collection of the random variables $\{f(\mathbf{x}_i), \mathbf{x}_i \in \mathscr{R}\}$ is normally distributed. A GP is fully specified by its mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$. Mathematically, we denote the process by:

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \tag{4.1}$$

In our regression task, the input $\mathbf{x}$ is the 120 D vector of the eye image and the output is the scalar rotation angles β and γ. For the following analysis we will use the horizontal rotation angle β for illustration. We have the model:

$$\beta = f(\mathbf{x}) + \varepsilon \tag{4.2}$$

where $\varepsilon \sim \mathscr{N}(0, \sigma_n^2)$ is the noise with variance σ_n^2. For training the GP regressor, we optimise the log marginal likelihood with respect to the parameters of the kernel. We denote the stacked vectors of training inputs $\mathbf{x}$ by $\mathbf{X}$ and the stacked vectors of training outputs β by B. Observing

that $B \sim \mathcal{N}(0, K(\mathbf{X},\mathbf{X}) + \boldsymbol{\sigma_n^2} I)$, the governing form of the marginal likelihood is

$$\log p(B|\mathbf{X}) = -\frac{1}{2} B^T (K(\mathbf{X},\mathbf{X}) + \boldsymbol{\sigma_n^2} I)^{-1} - \frac{1}{2}\log|K(\mathbf{X},\mathbf{X}) + \boldsymbol{\sigma_n^2} I| - \frac{N}{2}\log(2\pi). \tag{4.3}$$

where $K(\mathbf{X},\mathbf{X})$ is the covariance matrix evaluated between all pairs of training inputs. Note that these optimised parameters are used for inference later. More training details can be found in Appendix B.

At inference time, to obtain the testing outputs B_* for a set of testing inputs $\mathbf{X}_*$, we first write the joint distribution of B and B_* as:

$$\begin{bmatrix} B \\ B_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(\mathbf{X},\mathbf{X}) + \sigma_n^2 I & K(\mathbf{X},\mathbf{X}_*) \\ K(\mathbf{X}_*,\mathbf{X}) & K(\mathbf{X}_*,\mathbf{X}_*) \end{bmatrix}\right). \tag{4.4}$$

where $K(\mathbf{X}_*,\mathbf{X})$ is the covariance matrix evaluated between all pairs of testing inputs and training inputs and $K(\mathbf{X}_*,\mathbf{X}_*)$ is the covariance matrix evaluated between all pairs of testing inputs. The mean of the predictive distribution can thus be obtained as:

$$\begin{aligned} B_* &= \mathbb{E}[B_*|\mathbf{X}, B, \mathbf{X}_*] \\ &= K(\mathbf{X}_*,\mathbf{X})[K(\mathbf{X},\mathbf{X}) + \sigma_n^2 I]^{-1} B. \end{aligned} \tag{4.5}$$

And the variance is given by:

$$\mathbf{cov}(B_*) = K(\mathbf{X}_*,\mathbf{X}_*) - K(\mathbf{X}_*,\mathbf{X})[K(\mathbf{X},\mathbf{X}) + \sigma_n^2 I]^{-1} K(\mathbf{X},\mathbf{X}_*). \tag{4.6}$$

4.2 Kernels

We now briefly describe the structure for three kernels that we will test: the squared exponential (SE), the rational quadratic (RQ) and a novel proposed kernel named the separable smoothing (SS) kernel.

4.2.1 SE Kernel

The SE kernel has the form:

$$k(\mathbf{x_i},\mathbf{x_j}) = \sigma_s^2 \exp\left(-\frac{1}{2Nl^2}||\mathbf{x_i} - \mathbf{x_j}||^2\right) + \mathbb{1}_{i=j}\sigma_n^2 \tag{4.7}$$

where σ_s^2 is the scaling variance, l is the length-scale, σ_n^2 is the noise variance and N is the size of the feature vector, which is 120 in our case. Note that we now directly incorporate the noise term σ_n^2 in Equation 4.2 in the kernel structure.

4.2.2 RQ Kernel

The second kernel that we experiment with is the rational quadratic (RQ) kernel, which can be viewed as a scale mixture (an infinite sum) of SE kernels with different characteristic length-scales. The RQ kernel has the structure:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \sigma_s^2 \left(1 + \frac{||\mathbf{x_i} - \mathbf{x_j}||^2}{2\alpha N l^2}\right)^{-\alpha} + \mathbb{1}_{i=j}\sigma_n^2 \tag{4.8}$$

where σ_s^2 is the scaling variance, N is the size of the feature vector ($N = 120$), l is the length-scale, α is the scale mixture parameter, and σ_n^2 is the noise variance.

4.2.3 SS Kernel

We would also like to exploit the inherent structure of the two-dimensional (2D) image input data, which has a width of 10 pixels and a height of 6 pixels. Here we propose the separable smoothing kernel, which has the form:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \sigma_s^2 \exp\left(-\frac{1}{4NM^2}\left((\mathbf{x_i^L} - \mathbf{x_j^L})^T C(\mathbf{x_i^L} - \mathbf{x_j^L})) + (\mathbf{x_i^R} - \mathbf{x_j^R})^T C(\mathbf{x_i^R} - \mathbf{x_j^R})\right)\right) + \mathbb{1}_{i=j}\sigma_n^2 \tag{4.9}$$

where M is the pixel scale, N is the feature dimension ($N = 120$), x_i^{L}, x_j^{L} are two input data points for the left eye, x_i^{R}, x_j^{R} are two input data points for the right eye, and C is a covariance matrix. We design C to be the Kronecker product of two Toeplitz matrices C^y and C^x, that is

$$C = C^y \otimes C^x \tag{4.10}$$

where

$$C_{ij}^x = \exp\left(-\frac{(i-j)^2}{2l_x^2}\right) \tag{4.11}$$

and similarly for C^y. The kernel has five free parameters: scaling variance σ_s^2, pixel scale M, length-scales l_x and l_y and noise variance σ_n^2. The Kronecker product operation accounts for the name "separable" as we separate the pixels in the horizontal and vertical directions.

The Toeplitz matrices C^y and C^x are symmetric exponentiated distance matrices, where the value of the element in the matrix is large if $|i-j|$ is small and vice versa. The Toeplitz matrices have been widely used as a general, linear time-invariant low-pass filter [28, 2]. This is because if we perform a single-value decomposition (SVD) of a circulant Toeplitz matrix C [1]:

$$C = USU^H \tag{4.12}$$

[1] Circulant matrices have row vectors that are composed of the same elements and each row vector is rotated one element to the right relative to the preceding row vector. Though C^x and C^y are not strictly circulant, they can be approximated as circulant matrices in this case, hence their eigenvectors are approximately the Fourier modes.

The normalised eigenvectors (columns of U) of C are the Fourier modes ranked from low spatial frequency to high spatial frequency. The singular values (diagonals of S) decay very fast. In other words, high frequency Fourier modes are associated with small eigenvalues and low frequency Fourier modes are associated with large eigenvalues. Hence if we multiply a vector (e.g. $(\mathbf{x}_\mathbf{i}^\mathrm{L} - \mathbf{x}_\mathbf{j}^\mathrm{L})$) with C^x, $(\mathbf{x}_\mathbf{i}^\mathrm{L} - \mathbf{x}_\mathbf{j}^\mathrm{L})$ is decomposed into various frequency components, and the high frequency components (which are assumed to be noise) are very strongly attenuated but the low frequency components remain, hence the name "smoothing" kernel.

Note that in this case we separate the left eye image and the right eye image (i.e. $\mathbf{x}_\mathbf{i}^\mathrm{L}$ in Equation 4.9 now has dimension 10 by 6). This is because if we concatenate the images for the left and right eyes, we would inevitably have a discontinuity in our data, which violates the assumption for C^y.

4.3 Data Collection

We design the following data collection method to evaluate the performance of the three kernels. After the length scale and blind spot calibration (see Section §3.1; the experimental setup is explained in Section §2.2.1) we would obtain the three measurements needed to calculate the relative position of the user with respect to the screen (d, x_d and y_d). A red dot will then appear on the screen at a random location and the participant is asked to click the red dot continuously five times until the dot disappears. The participant is also specifically instructed to stare at the red dot during the clicks. As soon as the dot has disappeared a new red dot appears at another random location. We record (1) the participant's eye images and (2) the positions of the click which are assumed to align with the gaze locations x_g and y_g. We then perform simple trigonometric calculations similar to those in Section §3.1.2 to obtain the angles β and γ. These are the ground truth eye features and angles.

It is important to note that in the original WebGazer implementation, only points on the *inner* edges of the eye image were identified from Facemesh and subsequently used to crop out the eye image. However, in the first round of our implementation, we notice that the predictions in the vertical direction are much more volatile than in the horizontal direction. One factor that potentially contributes to this phenomenon is that when we look from left to right, the only body part that moves is the eyeballs. However, when we look from up to down, it is most likely that our eyelids will move as well. Hence the eyelid movements will potentially provide us with information on the eye rotation in the vertical direction and should be factored in. Thus in our implementation, we identify the points on the *outer* edges of the eye from Facemesh to crop out the eye image. We hope that the information embedded in the movements of the eyelids can be implicitly made use of by the regressor. The implementation details can be found in Section §A.2 in Appendix A.

We have collected data from a total of three participants, who will be referred to as Participant A, B and C, with 680 pairs of data from each. 500 pairs are used for training and 180 for testing in all following experiments. We also merge all data points and form a complete data set of size

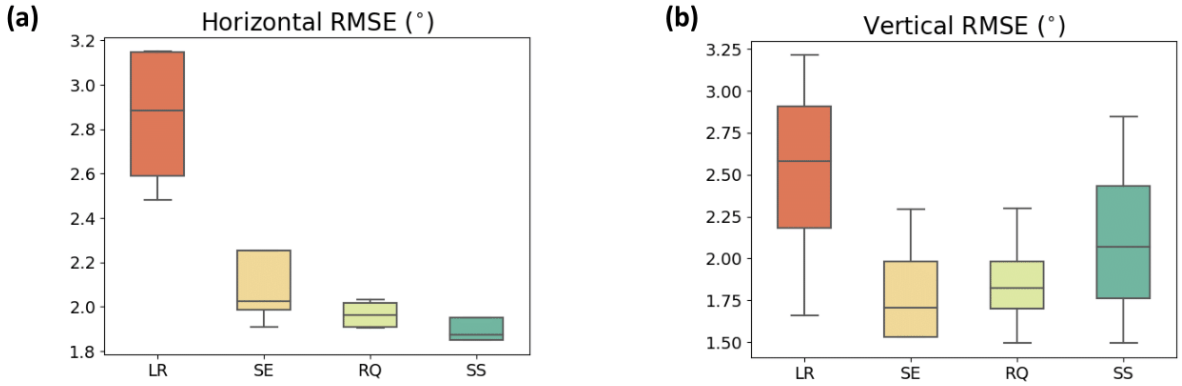


Fig. 4.1 Box plots for the root mean squared error (RMSE) for linear regressor, GP regressor with SE, RQ and SS kernels for (a) horizontal and (b) vertical angles for all participants. Note that all three GP kernels outperform the LR. However, no kernel outperforms any other significantly.

2040 named Participants All, where 1500 pairs are used for training and 540 pairs for testing. The complete information on Participants A to C is shown in Table A.1 in Appendix A.

4.4 Results

We now report the training and cross-validation results using (1) the mean absolute error (MAE) and (2) the root mean squared error (RMSE). Formally, given the ground-truth value of the angles β and γ, and the prediction results denoted as $\hat{\beta}$ and $\hat{\gamma}$, horizontal and vertical MAE is defined as

$$\begin{aligned} \text{Horizontal MAE} &= \frac{1}{N}\sum_{i=1}^{N}|\beta-\hat{\beta}|, \\ \text{Vertical MAE} &= \frac{1}{N}\sum_{i=1}^{N}|\gamma-\hat{\gamma}| \end{aligned} \tag{4.13}$$

where N is the total number of data points for testing. The horizontal and vertical RMSE is defined as

$$\begin{aligned} \text{Horizontal RMSE} &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\beta-\hat{\beta})^2}, \\ \text{Vertical RMSE} &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\gamma-\hat{\gamma})^2}. \end{aligned} \tag{4.14}$$

4.4.1 Training Results

The horizontal and vertical MAE and RMSE are reported in Table 4.1 and the box plots for RMSE are shown in Figure 4.1 for all three kernels. As a baseline, we also train a simple linear regressor (LR) (as in the original WebGazer package) and the results are reported in the same

Table 4.1 GP mean absolute error (MAE) and root mean squared error (RMSE) with linear regressor (LR) and GP regressor with SE, RQ and SS kernels. Best results are bold in text. Note that the GP regression results are better than the linear regression results. The experimental setup is explained in Section §2.2.1.

		Horizontal		**Vertical**	
		MAE (°)	**RMSE (°)**	**MAE (°)**	**RMSE (°)**
Participant A	LR	2.074	2.627	1.352	1.661
	SE	**1.559**	**2.033**	1.262	1.530
	RQ	1.559	2.033	1.233	1.495
	SS	1.636	2.102	**1.208**	**1.493**
Participant B	LR	1.845	3.150	1.719	3.214
	SE	**1.467**	**1.909**	**1.352**	**1.530**
	RQ	1.467	1.911	1.352	1.770
	SS	1.472	1.849	1.415	1.848
Participant C	LR	2.108	3.145	2.034	2.810
	SE	1.495	2.013	1.696	2.293
	RQ	1.495	2.013	**1.656**	**2.248**
	SS	**1.421**	**1.899**	1.690	2.293
Participants All	LR	1.891	2.481	1.816	2.352
	SE	1.427	1.910	1.432	1.876
	RQ	1.415	1.906	1.432	1.875
	SS	**1.387**	**1.849**	**1.415**	**1.848**

table. From the table and the plot, it is clear that all three kernels outperform the linear regressor in terms of both MAE and RMSE. Between the three GP kernels no kernel achieves significantly better results than any other. Since the SE kernel is computationally the cheapest at both training and inference time, as explained in Section §B.2, we prefer the SE kernel overall. The MAE is 1.43° for the SE kernel for the Participants All, which is a 22.9% improvement from the linear regressor with an MAE of 1.85°.

A plot of the predictions against true values for the horizontal rotation angle β and vertical rotation angle γ for data from Participant B are shown in Figure 4.2 for illustration for the SE kernel. Note that although the MAE and RMSE for γ is reported to be smaller than β, γ also spans a smaller range (between $-10°$ and $10°$) compared to β (between $-25°$ and $15°$). From Figure 4.2 we can see that the predictions for γ are actually slightly worse compared to β. We argue that the slightly worse performance in vertical angle prediction is attributed to the fact that the screen height (1080 px) is much smaller than the screen width (1920 px), leading to a lower signal to noise ratio (SNR) in the vertical direction compared to the horizontal direction.

Another important finding is that the presence of spectacles does not seem to have a negative impact on the accuracy of the results obtained, as the GP regressor seems to perform equally well irrespective of whether the participant is wearing glasses. This is contrary to the case for traditional infrared eye trackers where glasses can have an adverse impact on gaze estimation due to strong reflections and distortions they may cause [10, 34, 15]. In our experiment, though

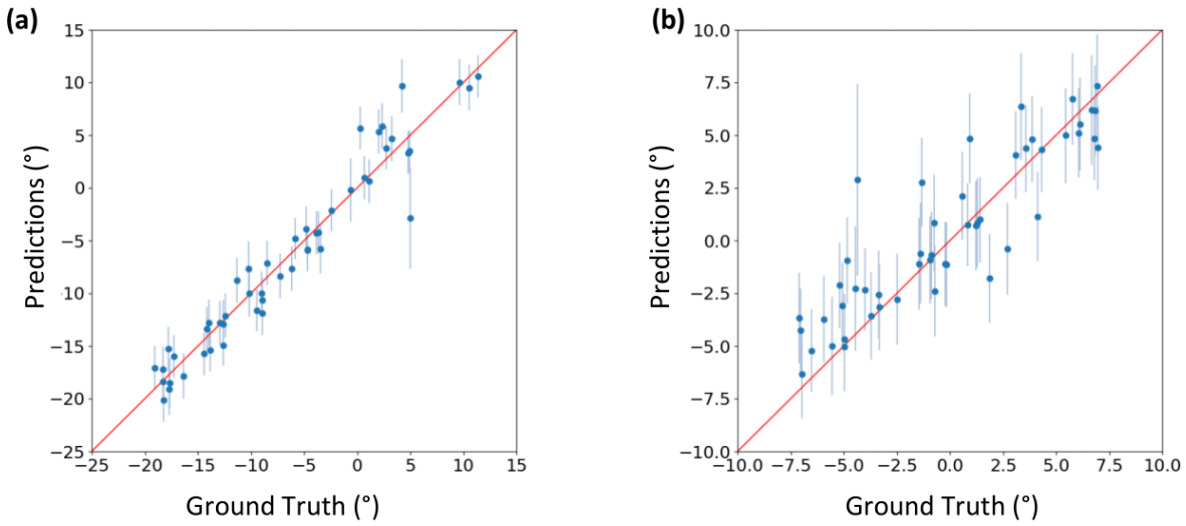


Fig. 4.2 Predicted against true values for (a) horizontal and (b) vertical angles for Participant B with SE kernel. Note that the red line represents the line with slope one and the error bars represent the posterior standard deviation. It can be seen that while predictions are relatively accurate for both cases, the predictions for vertical angles are still worse than those for horizontal angles as the predictions deviate more from the ground truth values. In addition, the posterior standard deviation is also larger for vertical angles compared with horizontal angles, indicating higher uncertainty in prediction.

Participant A and B are wearing glasses, the performance of the GP regressor for these two participants is not lower than Participant C, who is not wearing glasses. We argue that this is because any light reflections or distortions to eye shapes are accurately implicitly picked up by the GP regressor in its training process, so these factors have been accounted for in the predictions. This demonstrates that the GP regressor generalises well to participants' eye features irrespective of the presence of glasses. As a final note, the optimised hyperparameters are presented in Appendix C.

4.4.2 Cross-Validation Results

If the kernel parameters that we have learned from training from one set of data points are applicable to other sets of data points, then in actual implementation, after the calibration phase, we can directly predict the rotation angles from the eye images with any other user using Equation 4.5 without having to train the parameters for the new test user. We now perform cross-validation with the three GP kernels to see whether the parameters learned from one participant are applicable to another. Here we discuss four cases: Case (1): Other-Self, Case (2): Other-Other, Case (3): All-Self and Case (4): All-All. The name in front of the hyphen "-" refers to the data set from whom the parameters are learned and the name after the hyphen refers to the data set on whom the predictive distribution is conditioned ($\mathbf{X}$ in Equation 4.5). For example, Case (1) refers to the scenario where we use parameters learned from another user (someone different from the test user) and condition on the data set from the test user.

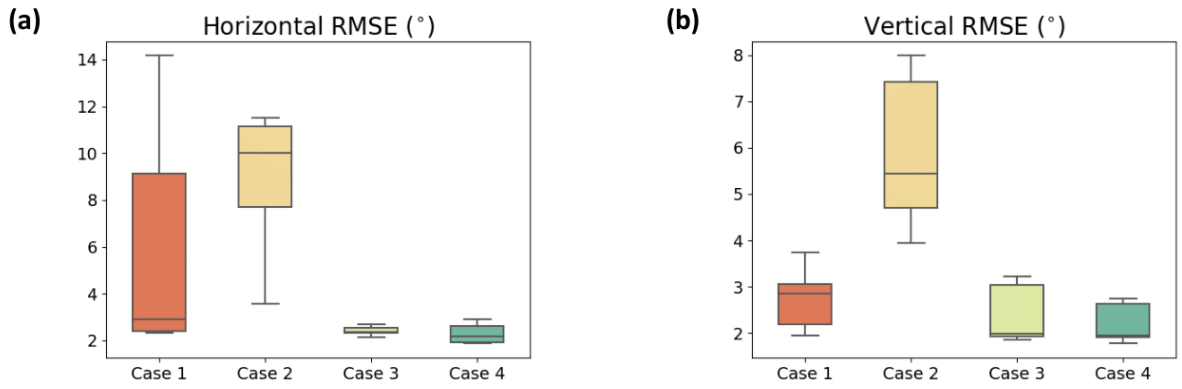


Fig. 4.3 Box plots for the root mean squared error (RMSE) for all four cross validation cases for (a) horizontal and (b) vertical angles for all participants. Case (3): All-Self and Case (4): All-All give the minimum RMSE overall.

For Case (2):Other-Other and Case (4):All-All since we are conditioning on a pre-determined data set (different from the test user), we can pre-compute matrix $\mathbf{X}$ in Equation 4.5 and its inverse so during actual implementation we only need to compute terms involving $\mathbf{X}_*$. The computational cost at inference time can thus be brought down from $\mathbf{O}(N^3)$ (for $[K(\mathbf{X},\mathbf{X})+\sigma_n^2 I]^{-1}$ term in Equation 4.5) to $\mathbf{O}(N)$ (for $K(\mathbf{X}_*,\mathbf{X})$ term in Equation 4.5), where N is the size of the training set. Note that since we are computing the eye rotation angles β and γ in real time, the size of the test dataset ($\mathbf{X}_*$ in Equation 4.5) is one. Secondly, this approach forgoes the need for the points clicking calibration exercise (see Section 2.2) to collect data points to be conditioned on since we are directly using the pre-computed data, thus simplifying the process even further.

The results are shown in Table 4.2, 4.3 and 4.4 for the SE kernel, the RQ kernel and the SS kernel respectively. We also plot the horizontal and vertical RMSE for all participants in Figure 4.3 for all four cases. For all three kernels, the best results are obtained for Case (4): All-All. One possible explanation is that the data set combining data from all three participants (Participants All) is able to cover different parts of the feature space. Hence when a new data point comes in, this new data point is sufficiently represented by the data set from Participants All and the kernel parameters learned from Participants All are applicable. Results obtained for Case (3): All-Self are only slightly worse than those obtained with Case (4): All-All so is also worth considering implementing. The worst results are obtained for Case (2): Other-Other. This leads to a very important insight: while cross-subject generalisation does not yield satisfactory results, generalising on the complete data set obtained from several participants achieves fairly good accuracy.

As an illustration we also present the plot of predictions against ground truth for Participant B for the SE kernel in Figure 4.4. The plots agree well with the average errors and RMSE results presented in Table 4.2, where Case (4): All-All yields the best agreement between ground truth and predictions and Case (2): Other-Other yields the worst agreement.

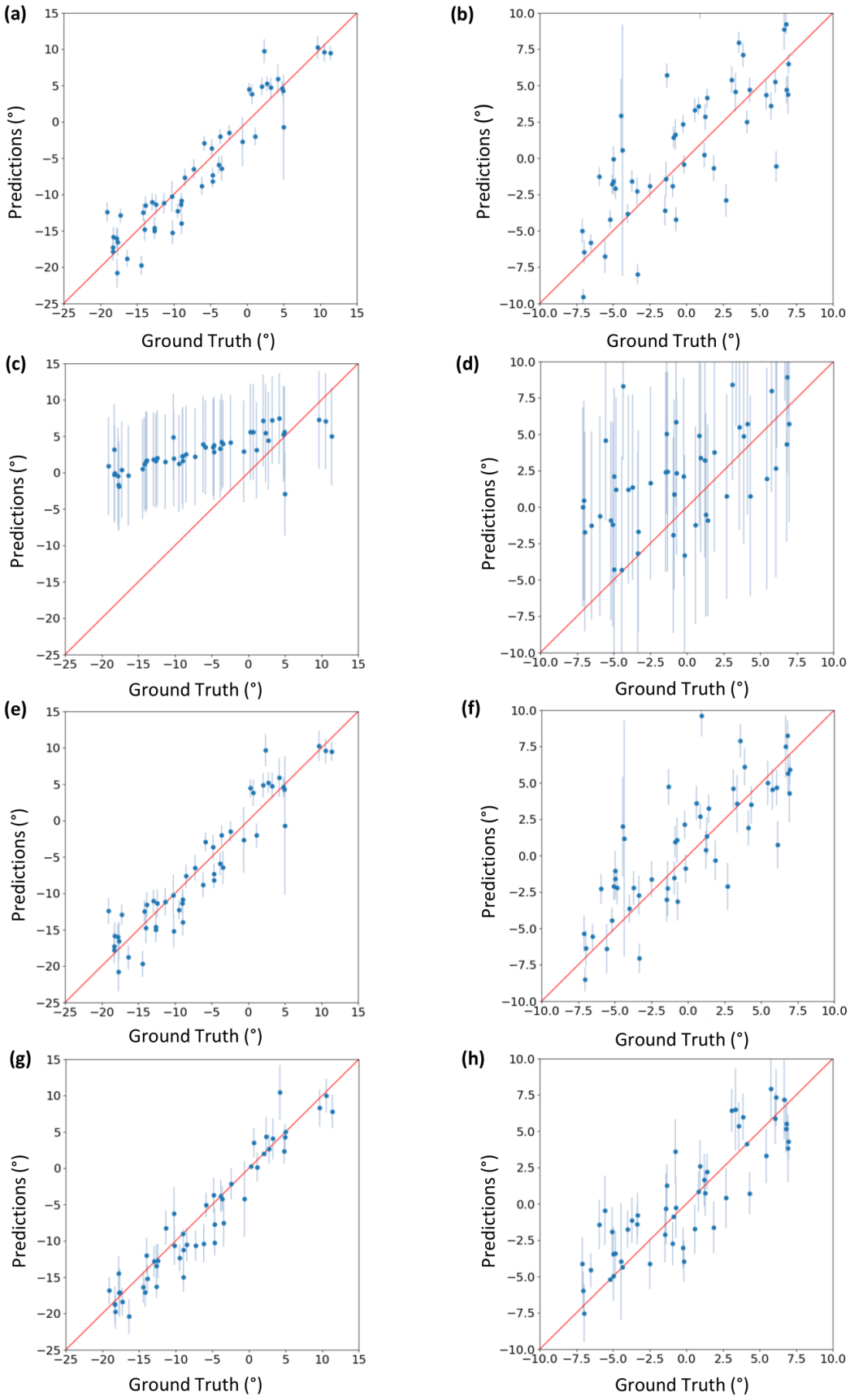


Fig. 4.4 Cross validation results for horizontal (left panel) and vertical (left panel) with data from Participant B and SE kernel. (a),(b): Case (1): Other-Self. (c),(d): Case (2): Other-Other. (e),(f): Case (3): All-Self. (g),(h): Case (4): All-All. Note that with SE kernel, Case (2) achieves the worst performance while Case (4) achieves the best performance.

Table 4.2 GP cross validation results with SE kernel. "**Params From**" indicates the data set from whom the optimised parameters are learned. "**Condition On**" indicates the set of data that we condition on at inference time. Cases (1), (2), (3), and (4) are defined in Section §4.4.2. It is clear that the best results are achieved when inference is performed when we use parameters learned from Participants All and conditioning on data from Participants All, which corresponds to Case (4).

	Case	**Params From**	**Condition On**	**Horizontal**		**Vertical**	
				MAE (°)	**RMSE (°)**	**MAE (°)**	**RMSE (°)**
Participant A	3	All	A	1.662	2.327	1.539	1.906
	4	All	All	**1.637**	**2.183**	**1.383**	**1.932**
Participant B	1	A	B	1.845	2.361	1.524	1.974
	2	A	A	8.496	10.046	4.817	5.751
	3	All	B	1.847	2.361	1.525	1.984
	4	All	All	**1.273**	**1.868**	**1.297**	**1.907**
Participant C	1	A	C	1.916	**2.512**	2.145	3.110
	2	A	A	6.125	6.953	7.178	7.985
	3	All	C	1.940	2.544	2.163	3.147
	4	All	All	**1.666**	2.610	**1.828**	**2.715**

Table 4.3 GP cross validation results with RQ kernel. Cases (1), (2), (3), and (4) are defined in Section §4.4.2. The best results are achieved with Case (4): All-All.

	Case	**Params From**	**Condition On**	**Horizontal**		**Vertical**	
				MAE (°)	**RMSE (°)**	**MAE (°)**	**RMSE (°)**
Participant A	3	All	A	**1.607**	**2.122**	1.481	1.855
	4	All	All	1.683	2.399	**1.284**	**1.778**
Participant B	1	A	B	1.822	2.313	1.510	1.958
	2	A	A	8.356	9.965	4.293	5.146
	3	All	B	1.791	2.279	1.483	1.934
	4	All	All	**1.297**	**1.937**	**1.200**	**1.780**
Participant C	1	A	C	2.489	3.300	2.778	3.740
	2	A	A	6.804	3.582	8.126	4.568
	3	All	C	1.882	**2.457**	2.063	3.003
	4	All	All	**1.685**	2.641	**1.766**	**2.640**

Table 4.4 GP cross validation results with the SS kernel. Cases (1), (2), (3), and (4) are defined in Section §4.4.2. The best results are achieved with Case (4): All-All.

	Case	**Params From**	**Condition On**	**Horizontal**		**Vertical**	
				MAE (°)	**RMSE (°)**	**MAE (°)**	**RMSE (°)**
Participant A	3	All	A	1.793	2.349	1.575	**1.967**
	4	All	All	**1.450**	**2.103**	**1.449**	1.992
Participant B	1	A	B	11.157	14.168	2.224	2.831
	2	A	A	9.504	11.516	3.298	3.951
	3	All	B	2.046	2.683	1.617	2.051
	4	All	All	**1.257**	**1.914**	**1.362**	**1.954**
Participant C	1	A	C	7.480	11.063	2.041	2.879
	2	A	A	25.048	26.971	40.583	41.155
	3	All	C	2.122	2.939	2.213	3.227
	4	All	All	**1.827**	**2.918**	**1.824**	**2.746**

Chapter 5

Implementation

We now implement the GP regressor with SE, RQ and SS kernels in the modified WebGazer package for two scenarios. In both scenarios we set the hyperparameters as those learned from Participants All. In the first scenario we implement an additional points clicking calibration exercise (explained in Section §2.2) to collect the user data to be conditioned on. Gaze location predictions are therefore conditioned on this set of data from the test user. This corresponds to Case (3): All-Self in Section §4.4.2. In the second scenario we pre-compute the matrix $\mathbf{X}$ and its inverse in Equation 4.5 using data from Participants All and store them in the script. Gaze location predictions are therefore conditioned on this readily available pre-computed data [1]. This corresponds to Case (4): All-All in Section §4.4.2. An illustration of these two cases is shown in Figure 5.1. We proceed to present our implementation results before discussing the performance and additional features.

5.1 Results

5.1.1 Case (3): All-Self

The results for Case (3): All-Self are summarised in Table 5.1. The box plots for all five cases (WebGazer original implementation, linear regression, SE, RQ and SS kernels) are shown in Figure 5.2. Across the three kernels, the SS kernel achieves the best performance, followed by the SE kernel. However, it is also noticed that there is a slight delay in the prediction using the SS kernel due to its high computational cost, even if we are only conditioning on roughly 40 data points. In addition, the difference in performance between all three kernels is marginal and there is no significant advantage of one kernel over another. Hence we prefer the SE kernel over the other two kernels due to its computational advantages.

[1] After tests and trials we decide on a total of 500 data points to be conditioned on ($\mathbf{X}$ in Equation 4.5 has size 500) so that there is no delay in real-time predictions.

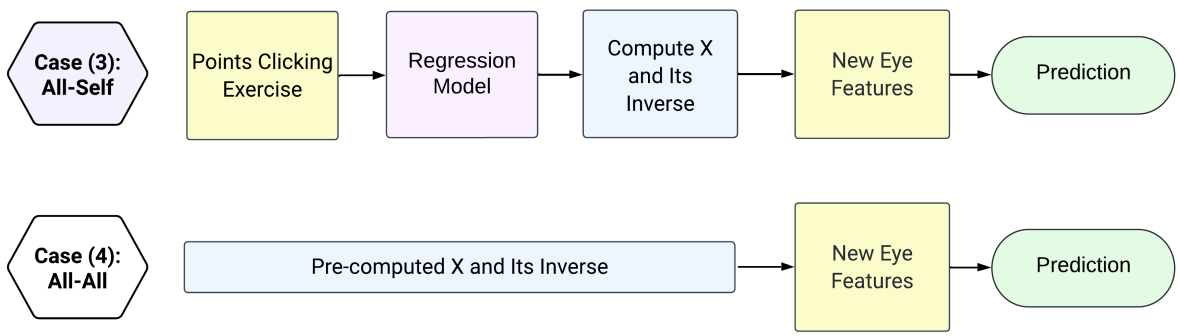


Fig. 5.1 Comparison in implementation of Case (3): All-Self and Case (4): Other-Self. Note that in Case (3): All-Self we need to carry out an additional points clicking exercise to generate data to be conditioned on but not in Case (3): All-All, since we are conditioning on pre-collected data.

Table 5.1 Results for proposed calibration phase with GP regression. Here we use the pre-determined optimised hyperparameters trained from Participants All and condition on current user data collected during calibration (Case (3): All-Self) and data collected from Participants All (Case (4): All-All). It is noticed that Case (4): All-All yields much worse results than Case (3): All-Self, indicating the former case is not viable. The definitions of metrics α_t, ε_x and ε_y and the experimental setup is explained in Section 2.2.1. For comparison of results obtained with the original WebGazer implementation and with linear regression refer to Table 2.1 and Table 3.1 respectively.

		Condition on User			**Condition on All**		
		α_{100} (%)	ε_x (px)	ε_y (px)	α_{100}	ε_x (px)	ε_y (px)
SE	Trial 1	6	140	33	0	132	261
	Trial 2	6	94	49	0	94	105
	Trial 3	8	85	50	0	64	102
	Trial 4	20	44	46	0	102	123
	Trial 5	8	91	38	0	106	149
	Average	10	**91**	43	0	100	148
RQ	Trial 1	2	74	110	0	568	32
	Trial 2	0	102	87	0	573	64
	Trial 3	2	142	62	0	640	174
	Trial 4	4	139	104	0	457	75
	Trial 5	12	109	96	0	482	89
	Average	4	113	92	0	544	87
SS	Trial 1	6	69	19	0	135	284
	Trial 2	6	93	32	0	161	285
	Trial 3	12	83	37	0	172	257
	Trial 4	14	101	46	0	147	199
	Trial 5	16	124	37	0	127	265
	Average	**11**	94	**34**	0	148	258

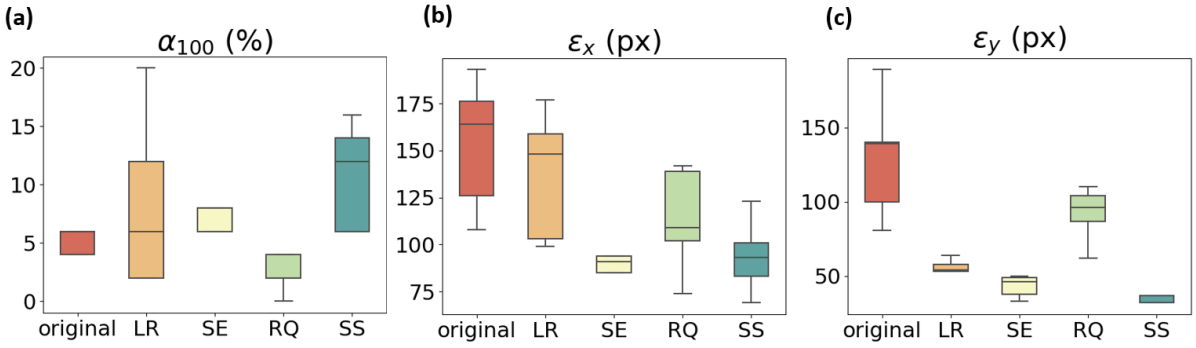


Fig. 5.2 Box plots for all five implementation cases (WebGazer original implementation, linear regression and SE, RQ and SS kernels with Case (3): All-Self (defined in Section §4.4.2)) for (a) α_{100}, (b) ε_x and (c) ε_y. The definitions of metrics α_w, ε_x and ε_y and the experimental set up is explained in Section §2.2.1. Note that the implementation with GP regression kernels outperforms both the original WebGazer and linear regression. Though the SS kernel achieves the overall best result, since the difference in performance is marginal, we still prefer the SE kernel overall due to its computational advantages.

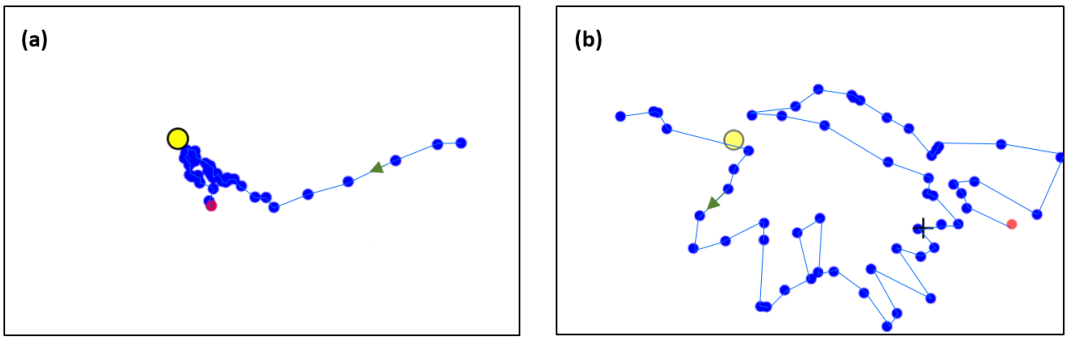


Fig. 5.3 Sample prediction trace for the quantitative accuracy test described in Section §2.2 for (a) GP regressor with SE kernel and (b) linear regressor for Case (3): All-Self. The yellow dot is the ground truth gaze location, the connected blue dots represent the prediction trajectory (the green arrow represents the trajectory direction), and the red dot is the prediction at the current time instant. Note that the linear regressor trace of predictions spans a wider part of the screen and is more volatile, while the GP regressor stabilises around the ground truth gaze location very quickly.

Comparing the SE kernel performance to the original WebGazer implementation results (Table 2.1), the accuracy has improved from 6% to 10%, ε_x is reduced from 153 px to 91 px (40.5%), and ε_y is reduced from 130 px to 43 px (66.9%). Comparing the SE kernel performance to the linear regression (LR) performance (Table 3.1), the accuracy has improved from 8% to 10%, ε_x is reduced from 137 px to 91 px, and ε_y is reduced from 58 px to 43 px. Apart from the improvement in accuracy and errors, the GP regressor is also preferred because it gives smaller fluctuations in predictions. An example of GP and linear regression prediction trace for the accuracy test described in Section §2.2 is given in Figure 5.3.

5.1.2 Case (4): All-All

The results for Case (4): All-All are also summarised in Table 5.1. It is surprising to note that the results turn out to be much worse than expected. The accuracy is consistently zero and ε_x for the RQ kernel and ε_y for the SS kernel are even larger than the errors obtained from the original WebGazer implementation (Table 2.1). Hence this case is not feasible for implementation.

5.2 Discussion

5.2.1 Performance

In Section §5.1.1 we observe that for Case (3): All-Self, the GP regressor with SE kernel has achieved significant improvement in accuracy compared to both the original WebGazer implementation and the linear regressor. We believe that this improvement in performance of our implementation is a result of three factors. Firstly, as explained in Section §4.3, the original WebGazer only crops out the *inner edges* of the eye images to form the feature set but in our implementation we crop out the *outer* edges of the eyes images to form the feature set to exploit information embedded in the movement of eyelids. This is, we believe, one of the most important factors that have contributed to the improvement in gaze location prediction especially in the vertical direction. Secondly, we introduced the improved measurement and calibration protocol (as explained in Section §3.1) to incorporate information on the user's relative position with respect to the screen into the algorithm. This introduces robustness against the movements of the user. Thirdly, instead of the linear ridge regression algorithm originally used in WebGazer, we now implement a Gaussian Process regressor to map from eye features to the rotation angles, which better models the non-linear relationship between the input eye features and output rotation angles.

In Section §5.1.2 we have concluded that the implementation results for Case (4): All-All is unsatisfactory. This is contrary to our finding in Section §4.4.2, where Case (4): All-All yields the best offline cross-validation results out of all four cases. We raise a possible explanation for this discrepancy. All three data sets collected in Section §4.4.2 are performed in one session for the same setting, while the tests performed on implementation are carried out much later. The changes in natural conditions (e.g. lighting conditions, positions of the desk and laptop etc) may have rendered the data collected in previous sessions unusable for the current session. Unfortunately due to time constraints, we do not have time to carry out experiments to test our hypotheses and account for the poor performance.

5.2.2 On-the-Go Calibration

The original WebGazer has an additional on-the-go calibration feature, which means that data points are collected *continuously* during the browser session whenever a user interaction occurs and the model is constantly updated. In our implementation, we stop data collection after the

points clicking calibration exercise for Case (3): All-Self. There are two reasons for this change in structure. Firstly, we notice that the assumption that the gaze locations align with the mouse click locations only holds true to a limited extent during user sessions. If the user is familiar enough with the web page, they can click on the point without specifically looking at that point. Hence, only data points collected during the points clicking calibration exercise where the user is specifically instructed to stare at the point they are clicking are deemed valid. Secondly, during implementation, we notice that the algorithm slows down significantly if we condition on a large number of data points (i.e. size of $\mathbf{X}$ in equation 4.5 is large), due to the expensive calculation of the $K(\mathbf{X}_*, \mathbf{X})$ term which is order $\mathbf{O}(N)$ where N is the size of $\mathbf{X}$. For real-time eye tracking the predicted gaze locations must be calculated fast enough such that there is no significant delay in painting the predicted gaze location on the screen. Hence we stop data collection after the calibration phase where we have collected roughly 40 data points for SE and RQ kernels. For the SS kernel the computational load is even heavier and we stop data collection after we have collected 20 data points during the calibration phase.

In the final version of our implementation of WebGazer we use the scenario for Case (3): All-Self, where optimised hyperparameters from Participants All are used and predictions are made while conditioning on data collected from the user during the points clicking exercise. As a final test, we play the two qualitative test games previously designed (see Section §2.2.2). It was noticed that the result is still far from satisfactory. The algorithm oftentimes predicts the square in which the gaze has fallen wrongly, which agrees with the low accuracy results reported in Table 5.1. It is clear that there is still much room for improvement for webcam-based eye trackers to achieve the level of accuracy necessary for the 4 by 4 navigation task that we designed as the benchmark.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this work we present an improved version of WebGazer, where we propose novel algorithms both at the calibration stage and the inference stage. The contribution of this project is three-fold. Firstly, we designed three accuracy tests to carry out a feasibility study to determine whether the accuracy of WebGazer is adequate for the benchmark navigation task. Upon recognising that the performance of the original WebGazer implementation is lacking, we proceeded to identify possible shortcomings and point out the most important factor, which is the fact that information on the user's relative location with respect to the screen is not taken into consideration. Secondly, we proposed our improved measurement and calibration protocol to measure the user's relative location with respect to the screen by implementing a simple card slider and blind spot calibration to gain knowledge of relevant environment variables such as the user's viewing distance with ideas inspired by human physiology. Thirdly, instead of the linear regressor implemented in the original WebGazer, we proposed a GP regressor to predict the user's eye rotation angles in horizontal and vertical directions from the eye features. Combining the angles with the user's viewing distance from the screen, we are able to calculate the gaze locations from simple geometrical relations.

An on-site data collection procedure was carried out, where we recruited three participants to collect data on their eye features and eye rotation angles. A total of 2040 pairs of data points were collected and analysed offline. The performance of a simple linear regressor is used as a baseline for our results. We tested the performance of three GP kernels: the SE kernel, the RQ kernel and a novel separable smoothing (SS) kernel. The mean absolute error and the root mean squared error between the model predictions and the ground-truth angles were used as metrics for comparison between models. We conclude that all three kernels achieve similar accuracy in performance but the SE kernel is preferred overall due to its low computational load. A comparison of the performance of kernels on data collected from different participants also suggests that factors such as the presence of spectacles do not have an impact on the performance of the regressor, demonstrating the robustness of our GP regressor against different traits of participants.

Another very important finding from cross-validation experiments is that the parameters learned from the data set of one participant generalises fairly well to other participants, and the best performance is achieved using parameters learned from the complete set of 2040 data points while conditioning on data collected from the test user (Case (3): All-Self in Section §4.4.2). This important insight allows us to perform inference directly during user browser sessions in an online fashion without the need for training, hence speeding up the prediction process. Finally, we implement the algorithm with Case (3): All-Self and test its performance. Compared to the original WebGazer, the accuracy has improved from 6% to 10%, ε_x is reduced from 153 px to 91 px (40.5%), and ε_y is reduced from 100 px to 43 px (66.9%), demonstrating the superiority of our approach.

6.2 Future Work

Despite a significant improvement in accuracy from the original WebGazer implementation, the performance of our algorithm is still limited compared to the most advanced infrared eye trackers available which are able to achieve an accuracy of 0.4° [3], which is equivalent to an error of less than 15 px on our setup. Despite the numerous measures we have taken to improve the performance of WebGazer, there are other factors that are worth considering, which we recommend as possible directions for future work.

Firstly, in the geometric calculations in Chapter 3 we have assumed that the head of the user is completely parallel to the screen surface, which does not necessarily hold true in real life. We can rotate our heads freely and this head rotation is not taken into consideration in the current implementation. One possible modification is to build a mapping model from the webcam images to the yaw, pitch and roll angles of the head using similar GP regression models. The incorporation of three-dimensional (3D) geometry into the model will make the eye tracker even more robust against changes in the user's posture.

Secondly, to realise real-time gaze prediction we choose to directly perform inference with kernel parameters pre-determined from offline training (Case (3): All-Self and Case (4): All-All). However, both approaches have their shortcomings: Case (3): All-Self only conditions on a small set of data points and results from Case (4): All-All proved to be unsatisfactory. One possibility is to combine these two approaches, where we perform sparse variational Gaussian process (SVGP) [13] to obtain a set of inducing points. Then, at the calibration stage, we utilise the smaller set of data collected from the current user to align the inducing points with predictions for the current user [18]. For a feasible implementation of SVGP it is also recommended to collect more data, which can be easily done via crowd-sourcing platforms.

Lastly, another extension worth exploring is the implementation of online sparse GP approximation, which dynamically updates the kernel parameters and the set of inducing points during user browser sessions instead of using a fixed set of pre-determined parameters and inducing points. However, we must emphasise that care must be taken to avoid expensive computation so that the eye tracker can run in real time.

References

[1] Armaly, M. (1969). The size and location of the normal blind spot. *Archives of Ophthalmology*, 81(2):192–201.

[2] Bäckström, T. (2013). Vandermonde factorization of toeplitz matrices and applications in filtering and warping. *IEEE Transactions on Signal Processing*, 61(24):6257–6263.

[3] Blignaut, P. and Wium, D. (2014). Eye-tracking data quality as affected by ethnicity and experimental design. *Behavior research methods*, 46:67–80.

[4] Brousseau, B., Rose, J., and Eizenman, M. (2020). Hybrid eye-tracking on a smartphone with cnn feature extraction and an infrared 3d model. *Sensors*, 20(2):543.

[5] Brown, R. J. and Thurmond, J. B. (1993). Preattentive and cognitive effects on perceptual completion at the blind spot. *Perception & Psychophysics*, 53:200–209.

[6] Burton, L., Albert, W., and Flynn, M. (2014). A comparison of the performance of webcam vs. infrared eye tracking technology. In *Proceedings of the human factors and ergonomics society annual meeting*, volume 58, pages 1437–1441. SAGE Publications Sage CA: Los Angeles, CA.

[7] Carter, B. T. and Luke, S. G. (2020). Best practices in eye tracking research. *International Journal of Psychophysiology*, 155:49–62.

[8] Degen, J., Kursat, L., and Leigh, D. D. (2021). Seeing is believing: testing an explicit linking assumption for visual world eye-tracking in psycholinguistics. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 43.

[9] Ehinger, B. V., Häusser, K., Ossandon, J. P., and König, P. (2017). Humans treat unreliable filled-in percepts as more real than veridical ones. *Elife*, 6:e21761.

[10] Gwon, S. Y., Cho, C. W., Lee, H. C., Lee, W. O., and Park, K. R. (2014). Gaze tracking system for user wearing glasses. *Sensors*, 14(2):2110–2134.

[11] Hansen, D. W. and Ji, Q. (2009). In the eye of the beholder: A survey of models for eyes and gaze. *IEEE transactions on pattern analysis and machine intelligence*, 32(3):478–500.

[12] Hansen, D. W. and Pece, A. E. (2005). Eye tracking in the wild. *Computer Vision and Image Understanding*, 98(1):155–181.

[13] Hensman, J., Fusi, N., and Lawrence, N. D. (2013). Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*.

[14] Hua, H., Krishnaswamy, P., and Rolland, J. P. (2006). Video-based eyetracking methods and algorithms in head-mounted displays. *Optics Express*, 14(10):4328–4350.

[15] Huang, Q., Veeraraghavan, A., and Sabharwal, A. (2017). Tabletgaze: dataset and analysis for unconstrained appearance-based gaze estimation in mobile tablets. *Machine Vision and Applications*, 28(5):445–461.

[16] Iacono, W. G. and Lykken, D. T. (1981). Two-year retest stability of eye tracking performance and a comparison of electro-oculographic and infrared recording techniques: Evidence of eeg in the electro-oculogram. *Psychophysiology*, 18(1):49–55.

[17] Jung, S.-G., Salminen, J., and Jansen, B. (2021). Implementing eye-tracking for persona analytics. In *ACM Symposium on Eye Tracking Research and Applications*, pages 1–4.

[18] Leroy, A., Latouche, P., Guedj, B., and Gey, S. (2022). Magma: inference and prediction using multi-task gaussian processes with common mean. *Machine Learning*, 111(5):1821–1849.

[19] Li, Q., Joo, S. J., Yeatman, J. D., and Reinecke, K. (2020). Controlling for participants' viewing distance in large-scale, psychophysical online experiments using a virtual chinrest. *Scientific reports*, 10(1):1–11.

[20] Papoutsaki, A., Sangkloy, P., Laskey, J., Daskalova, N., Huang, J., and Hays, J. (2016). Webgazer: Scalable webcam eye tracking using user interactions. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3839–3845. AAAI.

[21] Quen, M. T. Z., Mountstephens, J., Teh, Y. G., and Teo, J. (2021). Medical image interpretation training with a low-cost eye tracking and feedback system: A preliminary study. *Healthcare Technology Letters*, 8(4):97–103.

[22] Rayner, K. (2009). The 35th sir frederick bartlett lecture: Eye movements and attention in reading, scene perception, and visual search. *Quarterly journal of experimental psychology*, 62(8):1457–1506.

[23] Rohrschneider, K. (2004). Determination of the location of the fovea on the fundus. *Investigative ophthalmology & visual science*, 45(9):3257–3258.

[24] Rozsa, A. (2022). Using contingent praise to increase visual engagement in an asynchronous online learning environment: An eye tracking study.

[25] Safran, A. B., Mermillod, B., Mermoud, C., Weisse, C. D., and Desangles, D. (1993). Characteristic features of blind spot size and location, when evaluated with automated perimetry: Values obtained in normal subjects. *Neuro-ophthalmology*, 13(6):309–315.

[26] Saxena, S., Lange, E., and Fink, L. (2022). Towards efficient calibration for webcam eye-tracking in online experiments. In *2022 Symposium on Eye Tracking Research and Applications*, pages 1–7.

[27] Schmitt, K.-U., Muser, M. H., Lanz, C., Walz, F., and Schwarz, U. (2007). Comparing eye movements recorded by search coil and infrared eye tracking. *Journal of clinical monitoring and computing*, 21(1):49–53.

[28] Selesnick, I. (2017). Sparsity-assisted signal smoothing (revisited). In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4546–4550. IEEE.

[29] Steindorf, L. and Rummel, J. (2020). Do your eyes give you away? a validation study of eye-movement measures used as indicators for mindless reading. *Behavior research methods*, 52(1):162–176.

[30] Thomas, R. K. (1993). Introduction: A biopsychology festschrift in honor of lelon j. peacock. *Journal of General Psychology*, 120(1):5.

[31] Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA.

[32] Wolf, A. and Ueda, K. (2021). Contribution of eye-tracking to study cognitive impairments among clinical populations. *Frontiers in Psychology*, 12.

[33] Yang, X. and Krajbich, I. (2021). Webcam-based online eye-tracking for behavioral research. *Judgment and Decision Making*, 16(6):1485–1505.

[34] Zhang, X., Sugano, Y., and Bulling, A. (2019). Evaluation of appearance-based methods and implications for gaze-based applications. In *Proceedings of the 2019 CHI conference on human factors in computing systems*, pages 1–13.

[35] Zhu, Z., Ji, Q., Fujimura, K., and Lee, K. (2002). Combining kalman filtering and mean shift for real time eye tracking under active ir illumination. In *2002 International Conference on Pattern Recognition*, volume 4, pages 318–321. IEEE.

Appendix A

Information on Data Sets

A.1 Data Set Files

The files in the data set are in the format of NumPy arrays. There are three participants in total, namely Participant A, Participant B and Participant C. Their information is found in Table A.1. For each participant there are 10 files:

- **eye_features**: These are the 120-dimensional eye features from the original implementation cropped out using points on the inner arc of the eyes.
- **eye_features_new_arc**: These are the 120-dimensional eye features from the new implementation cropped out using points on the outer arc of the eyes. A more detailed explanation can be found in section A.2.
- **LPD**: Logical pixel density. This is the conversion factor from screen pixels to physical distances (millimeters).
- **horizontal_distances**: Horizontal distance from the left edge of the screen to the center of the eyes. The unit is pixels. This corresponds to x_d in Figure 3.3.
- **horizontal_positions**: x-coordinate of the gaze on the screen. The unit is pixels. This corresponds to g_x in Figure 3.1.
- **horizontal_angles**: Rotation angle of the eye in the horizontal direction. The unit is radians. This corresponds to β in Figure 3.1.
- **vertical_distances**: Vertical distance from the top edge of the screen to the center of the eyes. The unit is pixels. This corresponds to y_d in Figure 3.3.
- **vertical_positions**: y-coordinate of the gaze on the screen. The unit is pixels. This corresponds to g_y in Figure 3.1.
- **vertical_angles**: Rotation angle of the eye in the vertical direction. The unit is radians. This corresponds to γ in Figure 3.1.

Each file has 680 entries, where each entry corresponds to one data point. There is a one-to-one correspondence between entries in the files.

Table A.1 Description of Data Collected.

	No. of Data Points	**Eye Colour**	**Wears Glasses**	**Gender**	**Age**
Participant A	680	Brown	Yes	Female	23
Participant B	680	Brown	Yes	Male	23
Participant C	680	Brown	No	Male	21

A.2 Eye Image Cropping Implementation

In the original implementation by WebGazer, only points on the inner edges of the eyes are identified and subsequently used to crop out the eye image. Recognising the importance of information that the movement of eyelids potentially contains (as explained in more detail in Section §4.3), we identify points on the outer edges of the eye from Facemesh (shown in a mesh map provided on MediaPipe Facemesh github page) to crop out the eye image. The old and new key points used are summarised in Table A.2.

Table A.2 Description of old and new eye arc keypoints.

	Left Eye	**Right Eye**
Old (Top Arc)	25, 33, 246, 161, 160, 159, 158, 157, 173, 243	463, 398, 384, 385, 386, 387, 388, 466, 263, 255
Old (Bottom Arc)	25, 110, 24, 23, 22, 26, 112, 243	463, 341, 256, 252, 253, 254, 339, 255
New (Top Arc)	130, 247, 30, 29, 27, 28, 56, 190, 243	463, 414, 286, 258, 257, 259, 260, 167, 359
New (Bottom Arc)	130, 25, 110, 24, 23, 22, 26, 112, 243	463, 341, 256, 252, 253, 254, 339, 255, 359

Appendix B

Training Details

In Chapter 4 we train the dataset on three Gaussian Process (GP) kernels: the squared exponential (SE) kernel, the rational quadratic (RQ) kernel and the separable smoothing (SS) kernel. We now describe the training details and computational timings results

B.1 Kernel Derivative Calculations

During training, we optimise the log marginal likelihood with respect to the parameters of the kernel. For notational convenience we denote $K(\mathbf{X},\mathbf{X})$ as K. The governing form of the marginal likelihood is reproduced for convenience:

$$\log p(B|\mathbf{X}) = -\frac{1}{2}B^T(K+\sigma_{\mathbf{n}}^{\mathbf{2}}I)^{-1} - \frac{1}{2}\log|K+\sigma_{\mathbf{n}}^{\mathbf{2}}I| - \frac{N}{2}\log(2\pi). \tag{B.1}$$

Taking derivative with respect to parameters:

$$\frac{\partial}{\partial\theta_j}\log p(B|\mathbf{X},\theta) = \frac{1}{2}\mathrm{tr}\left((\alpha\alpha^{\mathbf{T}} - K^{-1})\frac{\partial K}{\partial\theta_j}\right) \tag{B.2}$$

where $\alpha = K^{-1}B$ [31]. We need to be able to calculate the derivative term, $\frac{\partial K}{\partial\theta_j}$. Here we present the derivative results for the three kernels.

B.1.1 SE Kernel

The form of the SE kernel is reproduced here:

$$k(\mathbf{x_i},\mathbf{x_j}) = \sigma_s^2\exp\left(-\frac{1}{2Nl^2}||\mathbf{x_i}-\mathbf{x_j}||^2\right) + \mathbb{1}_{i=j}\sigma_n^2. \tag{B.3}$$

The kernel has two parameters (ignoring the trivial noise term σ_n^2): the scaling variance σ_s^2 and the length-scale l. The derivative of the gram matrix ($\mathbf{K}$) with respect to each of the two

parameters is calculated as follows:

$$\frac{\partial K}{\partial \sigma_s} = 2\sigma_s \exp\left(-\frac{1}{2l^2 N}||\mathbf{x_i} - \mathbf{x_j}||^2\right) \tag{B.4}$$

$$\frac{\partial K_{ij}}{\partial l} = -\frac{1}{N} K_{ij} \frac{1}{l^3} \tag{B.5}$$

B.1.2 RQ Kernel

The RQ kernel equation is reproduced here:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \sigma_s^2 \left(1 + \frac{||\mathbf{x_i} - \mathbf{x_j}||^2}{2\alpha N l^2}\right)^{-\alpha} + \mathbb{1}_{i=j}\sigma_n^2. \tag{B.6}$$

The kernel has three parameters: the scaling variance σ_s^2, the length-scale l and the scale mixture parameter α. The derivative of the gram matrix with respect to each of the three parameters is calculated as follows.

$$\frac{\partial K}{\partial \sigma_s} = 2\sigma_s \left(1 + \frac{||\mathbf{x_i} - \mathbf{x_j}||^2}{2\alpha N l^2}\right)^{-\alpha} \tag{B.7}$$

$$\frac{\partial K_{ij}}{\partial l} = K_{ij} \left(1 + \frac{||\mathbf{x_i} - \mathbf{x_j}||^2}{2\alpha N l^2}\right)^{-1} \left(\frac{||\mathbf{x_i} - \mathbf{x_j}||^2}{N l^3}\right) \tag{B.8}$$

And

$$\frac{\partial K_{ij}}{\partial \alpha} = K_{ij} \left(\ln\left(1 + \frac{||\mathbf{x_i} - \mathbf{x_j}||^2}{2\alpha N l^2}\right) - \alpha \frac{||\mathbf{x_i} - \mathbf{x_j}||^2}{2\alpha N l^2 + ||\mathbf{x_i} - \mathbf{x_j}||^2}\right) \tag{B.9}$$

$$\frac{\partial K}{\partial \sigma_s} = 2\sigma_s \left(1 + \frac{||\mathbf{x_i} - \mathbf{x_j}||^2}{2\alpha N l^2}\right) \tag{B.10}$$

B.1.3 SS Kernel

The form of the SS kernel is reproduced here:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \sigma_s^2 \exp\left(-\frac{1}{4NM^2}\left((\mathbf{x_i^L} - \mathbf{x_j^L})^T C(\mathbf{x_i^L} - \mathbf{x_j^L})) + (\mathbf{x_i^R} - \mathbf{x_j^R})^T C(\mathbf{x_i^R} - \mathbf{x_j^R})\right)\right) + \mathbb{1}_{i=j}\sigma_n^2 \tag{B.11}$$

where C to is the Kronecker product of two Toeplitz matrices C^y and C^x, that is

$$C = C^y \otimes C^x \tag{B.12}$$

where

$$C_{ij}^x = \exp\left(-\frac{(i-j)^2}{2l_x^2}\right) \tag{B.13}$$

and similar for C^y. The kernel has 4 parameters: the scaling variance σ_s^2, two length-scales l_x and l_y, the pixel scale M. The derivative of the gram matrix with respect to each of the four

parameters is calculated as follows.

$$\frac{\partial K}{\partial \sigma_s} = 2\sigma_s \exp\left(-\frac{1}{4NM^2}\left((\mathbf{x_i^L}-\mathbf{x_j^L})^T C(\mathbf{x_i^L}-\mathbf{x_j^L})) + (\mathbf{x_i^R}-\mathbf{x_j^R})^T C(\mathbf{x_i^R}-\mathbf{x_j^R})\right)\right) \tag{B.14}$$

$$\frac{\partial K_{ij}}{\partial M} = -\frac{1}{2NM^3} K_{ij} \times \left((\mathbf{x_i^L}-\mathbf{x_j^L})^T C(\mathbf{x_i^L}-\mathbf{x_j^L})) + (\mathbf{x_i^R}-\mathbf{x_j^R})^T C(\mathbf{x_i^R}-\mathbf{x_j^R})\right) \tag{B.15}$$

$$\frac{\partial K_{ij}}{\partial l_x} = -\frac{1}{4NM^2} K_{ij} \times \left((\mathbf{x_i^L}-\mathbf{x_j^L})^T C^y \otimes \frac{\partial C^x}{\partial l_x}(\mathbf{x_i^L}-\mathbf{x_j^L})) + (\mathbf{x_i^R}-\mathbf{x_j^R})^T C^y \otimes \frac{\partial C^x}{\partial l_x}(\mathbf{x_i^R}-\mathbf{x_j^R})\right) \tag{B.16}$$

$$\left(\frac{\partial C^x}{\partial l_x}\right)_{ij} = \exp\left(-\frac{(i-j)^2}{2l_x^2}\right)\frac{(i-j)^2}{l_x^3} \tag{B.17}$$

where $\otimes$ denotes Kronecker product.

$$\frac{\partial K_{ij}}{\partial l_y} = -\frac{1}{4NM^2} K_{ij} \times \left((\mathbf{x_i^L}-\mathbf{x_j^L})^T \frac{\partial C^y}{\partial l_y} \otimes C^x(\mathbf{x_i^L}-\mathbf{x_j^L})) + (\mathbf{x_i^R}-\mathbf{x_j^R})^T \frac{\partial C^y}{\partial l_y} \otimes C^x(\mathbf{x_i^R}-\mathbf{x_j^R})\right) \tag{B.18}$$

where

$$\left(\frac{\partial C^y}{\partial l_y}\right)_{ij} = \exp\left(-\frac{(i-j)^2}{2l_y^2}\right)\frac{(i-j)^2}{l_y^3}. \tag{B.19}$$

B.2 Training

As described in Section §B.1, it is important that we calculate the derivative of the gram matrix with respect to each of the parameters, $\frac{\partial K}{\partial \theta_j}$ efficiently. Here we describe and compare two methods, namely analytical and numeric. For the analytical method we calculate the exact derivative terms $\frac{\partial K}{\partial \theta_j}$. For the numeric approach, we set an interval of $\Delta = 10^{-5}$ and the gradient for each parameter is calculated by taking the finite difference between the two kernel values:

$$\frac{\partial K}{\partial \theta_j} = \frac{K(\theta_j + \Delta) - K(\theta_j + \Delta)}{2\Delta} \tag{B.20}$$

The computational timings for these methods corresponding to the number of data points are summarised in Table B.1. The numeric method is actually more efficient than the analytical method for the SS kernel gradient calculations. This is most likely because the analytical method needs to keep track of many matrices whose size may be large hence taking up too much memory and thus slower. On the other hand, the SE and RQ training timings using the analytical method are much lower than the SS kernel due to their simpler structure and fewer free parameters. We must take note that since our aim is to implement a real-time eye tracker, the computational load must be small enough that calculations can be performed relatively quickly, ideally in milliseconds. The timings are recorded on a laptop with Windows 11 and Python 3.8. The laptop's processor is AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz and RAM is 8.00 GB.

Table B.1 Computational timings for the two gradient calculation methods: Analytical and Numeric for all three kernels (SE, RQ and SS).

No. of Data Points	**Analytical (SE)**	**Analytical (RQ)**	**Analytical (SS)**	**Numeric (SS)**
5	0.483	0.406	1.609	0.547
10	0.406	0.437	1.875	0.406
20	1.922	2.438	2.06	1.313
100	7.063	6.047	96.438	67.953

Appendix C

GP Regression Parameters

The optimised hyperparameters are presented in Table C.1, C.2 and C.3 for the squared exponential (SE), rational quadratic (RQ) and separable smoothing (SS) kernels respectively for training results in Chapter 4.

Table C.1 Optimised SE kernel parameters for different participants.

	Horizontal			**Vertical**		
	$\sigma_\mathbf{s}$	l	$\sigma_\mathbf{n}$	$\sigma_\mathbf{s}$	l	$\sigma_\mathbf{n}$
Participant A	0.130	64.4	0.001	0.214	117.7	0.001
Participant B	0.154	83.5	0.001	0.130	73.0	0.001
Participant C	0.153	45.1	0.001	0.119	44.9	0.001
Participants All	0.191	74.1	0.001	0.104	58.5	0.001

Table C.2 Optimised RQ kernel parameters for different participants.

	Horizontal				**Vertical**			
	$\sigma_\mathbf{s}$	l	α	$\sigma_\mathbf{n}$	$\sigma_\mathbf{s}$	l	α	$\sigma_\mathbf{n}$
Participant A	0.160	56.8	$1.47 \cdot 10^5$	0.001	0.171	95.5	$1.24 \cdot 10^4$	0.001
Participant B	0.238	98.3	$5.00 \cdot 10^{10}$	0.001	0.181	82.6	3033	0.001
Participant C	0.153	45.1	$2.69 \cdot 10^4$	0.001	0.223	87.3	0.256	0.001
Participants All	0.241	90.5	0.657	0.001	0.255	123.8	0.342	0.001

Table C.3 Optimised SS kernel parameters for different participants.

	Horizontal					**Vertical**				
	$\sigma_\mathbf{c}$	M	l_x	l_y	$\sigma_\mathbf{n}$	$\sigma_\mathbf{c}$	M	l_x	l_y	$\sigma_\mathbf{n}$
Participant A	0.155	35.3	4665.6	0.001	0.0122	0.236	103.9	1.40	0.83	0.001
Participant B	0.305	107.0	1.00	1.12	0.001	0.168	60.0	0.88	0.49	0.001
Participant C	0.172	40.4	0.56	1.25	0.001	0.129	37.2	0.81	0.22	0.001
Participants All	0.212	63.5	0.88	1.28	0.001	0.171	60.7	0.86	0.41	0.001

Appendix D

Risk Assessment Retrospective

The risks outlined in the risk assessment form at the start of the year are accurate. Given the computational nature of this project, the main risks are associated with the use of computers. I have diligently followed the guidelines and suggestions in the original risk assessment form, which have proven to be thorough and useful. I have taken note not to sit in front of the computer for more than eight hours a day and keep a good posture while working. I have also made sure to always take a break whenever I feel soreness in my eyes and only go back to work when my eyes feel fine. I would follow the same guidelines if I were to take another project of the same nature.