Gaze-Based Text Entry with Common RGB Cameras

by

Yuchi Chen

M.Sc., Beijing University of Posts and Telecommunications, 2015
B.Sc., Beijing University of Posts and Telecommunications, 2012

Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Spring 2024

Declaration of Committee

Name:	**Yuchi Chen**
Degree:	**Master of Science**
Thesis title:	**Gaze-Based Text Entry with Common RGB Cameras**
Committee:	**Chair:** Zhengjie Miao Assistant Professor, Computing Science
	Jiangchuan Liu Supervisor Professor, Computing Science
	Qianping Gu Committee Member Professor, Computing Science
	Jiannan Wang Examiner Associate Professor, Computing Science

Abstract

Gaze-based text entry has seen success in both academic researches and commercial products. However, such solutions typically involve specialized hardware, limiting their scalability and feasibility. Commercial gaze tracking solutions that use common RGB webcams yield coarse gaze regions, which are not directly ready for text entry tasks. In this project, to fill the open gap of the cost-efficient gaze-based text entry task with common RGB cameras, we propose a solution that leverages the advantage of both appearance-based and model-based methodologies of gaze tracking, and tackle challenges for the text entry task. With a prototype as the proof-of-concept, we show that our system can enable a user to conduct tap typing on a virtual keyboard with a gaze point on the screen, at a typing speed ranging from 18 to 30 letters per minute with an accuracy of over 90%.

Keywords: human-computer interaction; gaze tracking; text entry; machine learning

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Jiangchuan Liu for his great guidance and support. I would like to thank my lab mates and staff at the School of Computing Science for all their help and support throughout my studies at Simon Fraser University. I am very grateful to Dr. Qianping Gu, Dr. Jiannan Wang and Dr. Zhengjie Miao, for their collaboration and support as the members of my project committee.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Human gaze tracking has been a long lasting topic in the area of Human-Computer Interactions (HCI), and has seen commercial success in recent years. Particularly, a series academic and industrial efforts have been witnessed to explore gaze-based text entry as a part of novel HCI technologies, ranging from research projects like TAGSwipe [23], to cutting edge virtual/ augmented reality products like Apple Vision Pro[1]. Most of such efforts, however, typically involve specialized hardware, like the iMotions infrared eye tracker[2] in TAGSWipe, or the manufactured infrared camera in Apple's head mounted display (HMD), which can be too costly for widespread adoptions. Commercial products that work on common, low-cost RGB cameras tend to provide coarse gaze detection result that is not directly read for the text entry task[3]. That said, a cost-efficient solution to gaze-based text entry still remains an open gap.

As a crucial part of cost-efficient gaze tracking solutions, common RGB cameras are widely utilized in many research efforts. Generally, gaze tracking projects based on RGB cameras can be categorized into two categories: *model-based* and *appearance-based*. The main difference lies in how they proceed the image inputs of human's face and eyes, which determines their quality of delivery and computational costs. Early efforts are mostly model-based [13], since lightweight, cost-efficient solutions to human face and eye detection are rich in the area of conventional computer vision technologies; the gaze is then estimated with sophisticated algorithms, either by the shapes of the eyes and pupils [5] or the corneal reflection from them [30]. In light of outstanding advancement in machine learning, notably represented by deep learning technologies, later efforts trend appearance-based, an end-to-end methodology acknowledging the fact that the image of human face and eyes suffice to directly derive the gaze [46]. Comparing with those model-based, appearance-based solu-

[1]https://www.apple.com/apple-vision-pro/

[2]https://imotions.com/products/hardware/smi-redn/

[3]https://beam.eyeware.tech/

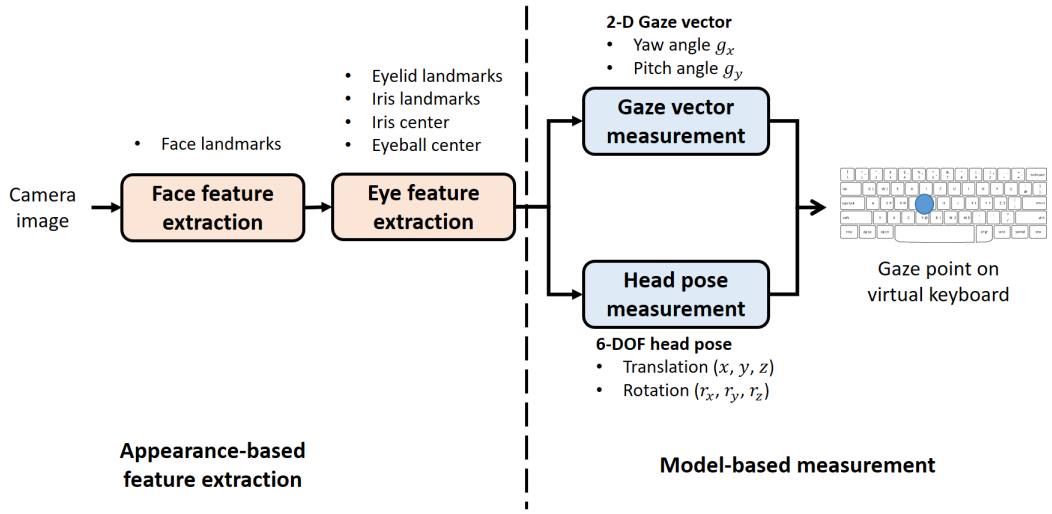


Figure 1.1: Overview of our gaze-based text entry solution, which consists of appearance-based feature extractions and model-based measurements, yields gaze point measures, and uses such measure as the hint for keystrokes on a virtual keyboard.

tions can be generally more robust against dynamic conditions, while preserving the quality of delivery. Such a superiority of cause comes with costs. For appearance-based solutions, with more designated objectives or higher accuracy or precision requirements, comes larger datasets or more complicated structure to build the neural network, which raises the overhead of both model training and inference, resulting in less scalability and flexibility than model-based solutions.

Inspired by the scalability of model-based solutions and the robustness of appearance-based solutions, in this project, we propose a solution that tries to combine them both for the task of realizing gaze-based text entry. Figure 1.1 illustrates the overview of our solution, consists of appearance-based feature extraction, including face and eye features, and model-based measurements, including gaze vector and head pose. Different from appearance-based solutions that yield results end-to-end, we use Deep Neural Networks (DNNs) to extract useful user features from images captured by RGB cameras, taking the advantage of the reliability of DNNs in human face and eye detection. Following a model-based principle thereby, we design subtle algorithms that can utilize the user features and derive the gaze point on the screen. To achieve the text entry, this gaze point measure is then used as the hint for the user to issue a keystroke on a virtual keyboard. Due to the fact that the gaze point measure can be unstable because of systematic source of error or user behaviors, we solve this practical challenge in our prototype implementation by measuring historical average and interpolating between adjacent measures.

In conclusion, our contribution is three-fold: (1) we tackle the challenge of cost-efficient solution to gaze-based text entry, which is based on common off-the-shelf RGB cameras, (2) we propose a solution that subtly combines appearance-based and model-based methodology to achieve gaze point tracking, (3) we optimize the gaze point measure for text entry task, and (4) we showcase our solution with a prototype that enables gaze-based text entry under a manner of tap typing.

Chapter 2

Literature Review

In this project, we focus on realizing the text entry task by tracking user gaze point on the target screen with common RGB cameras like webcams. Therefore, this work is closely related to computer vision-based gaze tracking technologies, the text entry methods, and the application of gaze tracking in text entry tasks in research literature.

2.1 Computer Vision-Based Gaze Tracking

During the past years, the mainstream of Computer Vision (CV) based gaze tracking technologies falls into two major categories: *model-based* [13] and *appearance-based* [46]. As they both take the images of users' faces and eyes as input, the difference mostly lies in the information extraction [37].

The model-based solutions tend to decode geometric features from the image input. For the gaze tracking task, model-based solutions typically resolve the user gaze vector, the indicator of the line-of-sight emitting from the user's iris, by deriving geometric features of eyes, such as the shapes of eyelids in the image input [5], or the corneal reflection in the iris [30]. Model-based gaze tracking is particularly popular in early research efforts, mostly due to mature, lightweight implementations of CV-based feature extraction algorithms, and the flexibility in the processing of geometric features. For example, EyeTab [40] extracted the user's iris contour from image input, and yielded the gaze point on a tablet by calculating the intersection between the gaze vector and the screen. As a later work, ScreenGlint [17] outperformed EyeTab by exploiting the *glint* of the screen on the user's cornea for gaze point detection. ScreenGlint applied a Gaussian regression process to establish the mapping between the features derived from eye images and a 4×7 grid of square cells on the screen. While ScreenGlint achieved a mean absolute error (MAE) of around 1.47 cm without head pose variations, it suffered from a MAE of 1.72 cm when head pose variation presents, revealing the importance of resolving the user's head pose in gaze tracking.

The appearance-based solutions, alternatively, do not decode geometric features from the image input; instead, appearance-based solutions leave this decoding task to machine

learning based models, and tend to derive the target information end-to-end, such as the user gaze vector or point on the screen, directly from the image input [36]. The advancing machine learning and deep learning technologies open the opportunity for appearance-based gaze tracking technologies to keep functional against various image input qualities, with the tradeoff between the robustness of achieving objectives, and the restricted flexibility and scalability caused by limitations of datasets. Holland *et al.* [16] presented an early neural network based solution to determine the end-to-end mapping between the eye image input and the gaze point on a commodity tablet. Huang *et al.* [18, 19] later presented a similar appearance-based solution for tablets, but utilized a Random Forest regression model and presented a dataset called TabletGaze. Different from these projects, GazeCapture [22] provided an appearance-based solution for both tablets and smartphones, with a dataset of mappings between user face images and 13 major gaze points, collected from over 1,450 people with a total of 2.5 million frames. In addition to this dataset, iTracker, a CNN-based end-to-end model is proposed for gaze point tracking, achieving an average prediction error of 2.53 cm and 2.12 cm, with and without calibration respectively. Based on both datasets of TabletGaze and GazeCapture, iMon [20], a latest appearance-based solution, improved previous efforts by constructing a probabilistic 2D heatmap gaze representation input and refined models to overcome various source of errors in the dataset, outperforming previous works by 22% to 28%.

In this project, we intend to take advantage of the upside of both model-based and appearance-based solutions. In particular, acknowledging the flexibility and scalability of geometric feature processing in model-based solutions, we do not try to derive the gaze point from the image input end-to-end, but the 3D location of the user's iris and 3D gaze vector from the image input, and geometrically calculate the gaze point on the surface of the target screen. On the other hand, while model-based solutions are built on conventional CV-based feature extraction algorithms, we utilize deep learning-based models for feature extraction, so as to achieve the robustness similar with appearance-based solutions.

2.2 Text Entry Methods

Text entry is a basic yet crucial task in human-computer interaction, thus it has long been a heated topic in HCI research area. While some research effort was put on hand writing [44], the majority of existing research works focus on the other two mainstream text entry methods: tap typing and word-gesture typing, also known as swipe typing. Tap typing follows conventional text entry rationale, that the user inputs a word letter by letter, where each letter is registered by a keystroke on a keyboard. Swipe typing, on the other hand, was proposed particularly after touch screen appeared, which significantly altered the text entry habits of mobile device users, as it could be challenging to preciously reach a virtual key on a small touch screen. Instead of inputting letter by letter, swipe typing typically allows

a user to swipe over the keyboard along a trajectory composed by letters in the designated word.

Both tap typing [45, 12] and swipe typing [26, 6] can be built on a conventional QWERTY keyboard, which is suitable for leveraging users' existing typing skills in various novel scenarios. Nevertheless, in some scenarios the implementation or usage of QWERTY keyboard can be challenging, thus the need for customized keyboard rises. For instance, to achieve an eyes-free implicit text entry method, finger limbs and tips are widely used as the surface of a virtual keyboard, which is typically customized in diverse ways [41, 27, 43]. Beside of these works, some other solutions were witnessed that either involve other keyboard locations [35], body parts like nose tips [21] and wrists [11], or novel input methods like phrase typing [42], an advanced swipe typing alters from word-gesture typing by directly enabling phrase input.

In this project, we intend to utilize the user gaze point on the target screen as the hint for text entry, which essentially requires a one-fingered text entry method. Therefore, we implement tap taping on QWERTY keyboard as a basic input method, and further explore and leverage existing swipe typing methods for better performance.

2.3 Gaze Tracking in Text Entry

Although eye tracking technologies have been long advancing for decades, the application of eye tracking technologies in HCI, and particularly for the text entry task, has been witnessed only in recent score. Drewes *et al.* [9] presented an early interaction system by utilizing the user gaze point and its movement. This solution relied on a third-party commodity eye tracker to track a user's eyes, whom was asked to stay still against a target Nokia N80 phone model. A series of works on gaze-based interaction, which mostly focused on text entry, were published ever since.

Few work developed customized text entry function without using a third-party eye tracker. For example, iType [25] proposed an appearance-based tap typing entry by tracking users' gaze point on a customized 4×3 grid keyboard on a smartphone, or a 5×4 grid keyboard on a tablet, using the front camera of the target device. While reaching the keystroke detection accuracy of 97% and 89% for static and dynamic scenarios respectively, iType suffered from a significant response delay of 364 ms, and a typing speed of around 0.5 character per second, due to the overhead of conducting end-to-end gaze point detection from the image input. Alternatively, acknowledging the challenge in realizing the gaze point tracking, many other research efforts tend to leverage third-party commodity eye trackers, and thus focus on achieving various objectives and improving the performance, such as accuracy and response, for text entry methods ranging from tap typing to swipe typing. For instance, Lu *et al.* [28] proposed GlanceType, a tap typing on split keyboards on a tablet, with a commodity Pupil-Labs eye tracker. Similarly, TAGSwipe [23] utilized a commodity

iMotions eye tracker to realize gaze-based swipe typing on a desktop, by asking the user to tap on a smartphone. As a popular manufacturer of commodity eye trackers, Tobii's eye tracker products were adopted in more research works [10, 34, 31, 24, 15, 8]; some also involve head mounted displays (HMD) to transfer the gaze information between the real world and AR or VR environments [33, 38, 14].

In this project, we aim at realizing the gaze tracking with common RGB cameras, thus essentially being different from the previous efforts that utilize third-party commodity eye trackers. In addition, in order to overcome the limitation brought by appearance-based gaze tracking solution in previous efforts like iType, we do not intend to reach the gaze tracking end-to-end from image output; instead, we tackle the challenge in resolving gaze point detection by geometric methods following a model-based principle.

Chapter 3

Gaze-Based Text Entry

As illustrated in Figure 1.1, our approach consists of appearance-based feature extraction and model-based measurement. In this chapter, we break down each part with details.

3.1 Appearance-Based Feature Extraction

For gaze point tracking task, the necessary features must be acquired from the user's face, which is captured by the RGB camera. For appearance-based methodology, the features are extracted directly from the raw image captured by the camera, without decoding any geometry information from the raw image. As such, the features include the face detection to start with. From the face image extracted from the raw image, the face features and eye features are extracted.

3.1.1 Face Detection

For face detection, we first applied a conventional CV technology called the Viola-Jones object detection algorithm, as a preliminary and lightweight method. This algorithm is essentially a cascaded classification on the Haar-like features derived from the captured image. The Haar-like features are the differences of average intensities between different regions in the image; as a result, these features include useful information for detecting and distinguishing among objects. A major issue is that the detection results highly depends on the image quality and the angular pose of the face in the image.

For more reliable detection in the proof-of-concept prototype, we utilize the Dlib library [1], a famous open-source project that implements both conventional CV algorithms and deep learning based algorithms. In particular, we choose the histogram of oriented gradients (HOG) and linear Support Vector Machine (SVM) face detector from the python-based Dlib for lightweight and accurate face detection.

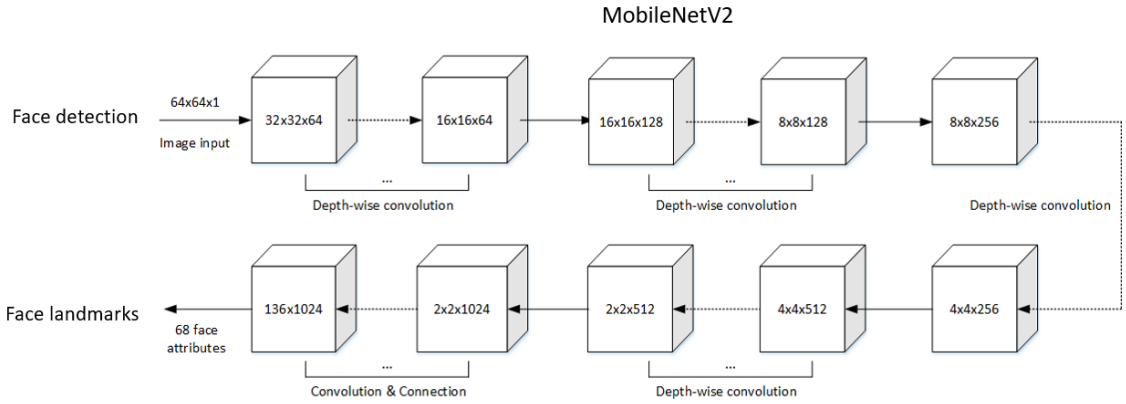


Figure 3.1: MobileNetV2 model for face feature extraction, including 68 face landmarks.

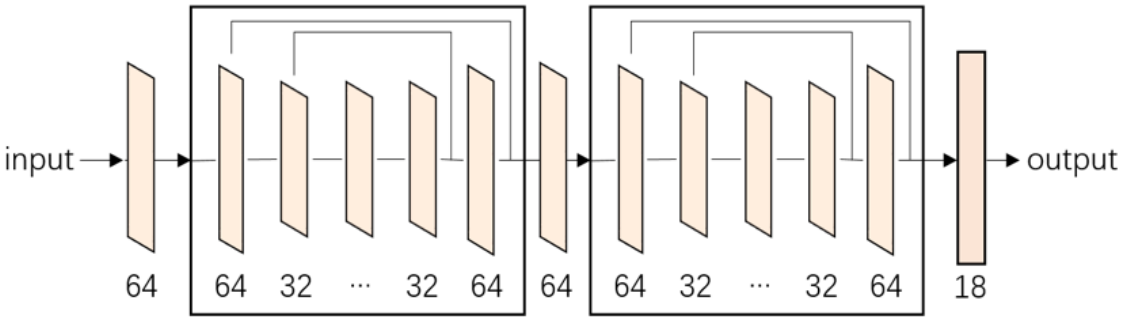


Figure 3.2: Hourglass-based model for eye feature extraction.

3.1.2 Face Feature Extraction

For reliable face landmark detection, we utilize a deep learning-based detection from the MobileNet V2 model. MobileNet V2 is a general-purpose, lightweight convolutional neural network (CNN) for object detection tasks.

The structure of our MobileNet V2 model is shown in Figure 3.1. Different from the CNNs that applies traditional convolution layers, the MobileNet V2 model splits the layer into two subtasks: a depth-wise convolution for filtering inputs, and a point-wise convolution for combination of the filtered values. The two subtasks together form a depth-wise separable convolution block, which operates faster than traditional convolution layers.

In our proof-of-concept prototype, we choose an open PyTorch-based pre-trained MobileNet V2 model [7]. The MobileNet V2 model we use takes one channel 64×64 image inputs and yields 68 face landmarks.

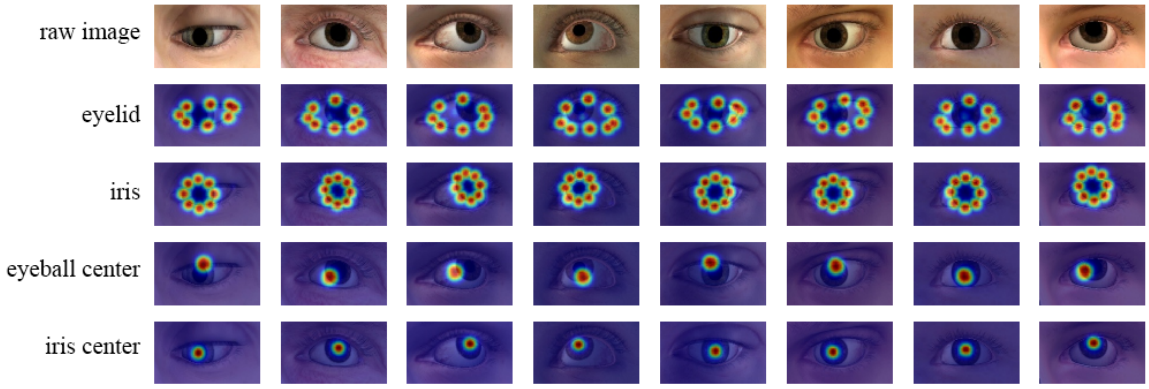


Figure 3.3: Synthesized eye images and the eye feature extraction results from the model, including eyelid landmarks, iris landmarks, eyeball center and iris center.

3.1.3 Eye Feature Extraction

For eye detection, we train an hourglass-based neural network model, which is inspired by GazeML [29], a representative appearance-based eye landmark detection project. As a tradeoff between the performance and computational costs, we customize the model to be two-layer hourglasses wrapped by full connection layers, as shown in Figure 3.2.

The model is trained with a synthesized eye image dataset from UnityEyes [39], a research project that can generate realistic eye images and provide corresponding parameters. We create 100,000 images of human eyes, with various skins, gaze directions and head poses. With the output of landmarks of eyelid and iris, and the implicit parameters like eyeball center, the hourglass-based model yields the 2D location of these landmarks on the input image, as shown in Figure 3.3. These landmarks serve as key inputs for the model-based gaze vector measurement in a similar way as face landmarks for head pose measurement. Some more quantitative details on the dataset are in Appendix A.

3.2 Model-Based Measurement

In this section, we introduce our model-based measurement on key parameters, include the head pose, the gaze vector, and eventually the gaze point [32].

3.2.1 Head Pose Measurement

The user's head pose includes two key features: the transform and rotation. The transform indicates the 3-D location of the user's head from the camera, including the Euler distance between the head and the screen. The rotation, on the other hand, indicates the direction to where the user's head is facing with, impacting the measurement of the user gaze vector, as shown in Figure 3.4.

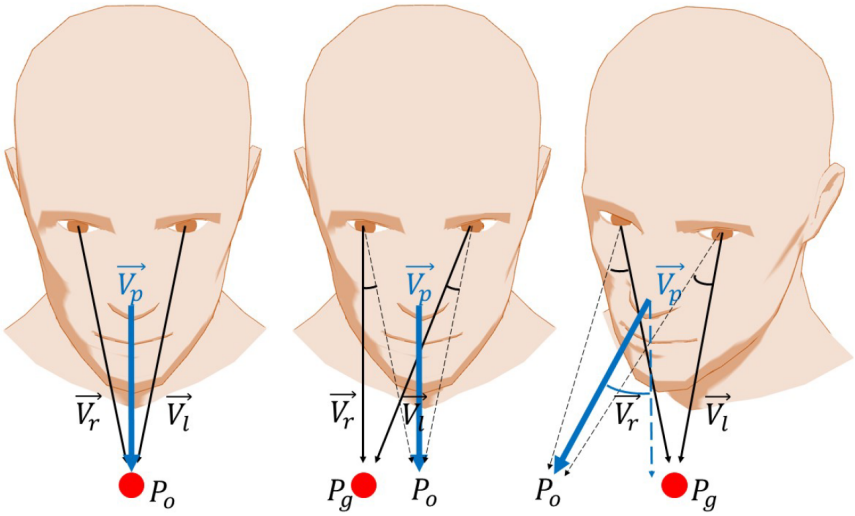


Figure 3.4: The rotation of the head impacts the measurement of the gaze vector. Point P_g is the intersection point of the two gaze vectors on the screen, and it can deviate from the point P_o, the default position, when the user is facing away from the screen.

For head pose measurement, we take the 3-D location of the face landmarks on a generic head model as a reference, and use the 2-D location of the user's face landmarks in the image to compute the 6 Degree of Freedom (6-DOF) head pose, i.e. the transform on three dimensions (x, y, z) and the rotation (α, β, γ) along the 3 dimensions. Specifically, denote matrices $\boldsymbol{R}_x(\alpha)$, $\boldsymbol{R}_y(\beta)$ and $\boldsymbol{R}_z(\gamma)$ as the rotation matrix on three dimensions, then the general rotation matrix $\boldsymbol{R}_r$ is calculated as follows:

$$\boldsymbol{R}_r = \boldsymbol{R}_z(\gamma)\boldsymbol{R}_y(\beta)\boldsymbol{R}_x(\alpha) \tag{3.1}$$

Denote the transform vector $T = (t_x, t_y, y_z)^T$, then the relationship between the 2D location of a face landmark (u, v) and its 3D location (x, y, z) is as follows:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & x_o \\ 0 & f_y & y_o \\ 0 & 0 & 1 \end{bmatrix} (\boldsymbol{R}_r|T) \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.2}$$

where s is the scale factor, f_x and f_y are the focus length of the front camera in terms of pixels, and (x_o, y_o) is the principle point, which is also the center of the screen. By solving this Perspective-N-Points (PNP) problem, the rotation matrix R_r and transform vector T can be calculated, and then the 6-DOF head pose are derived.

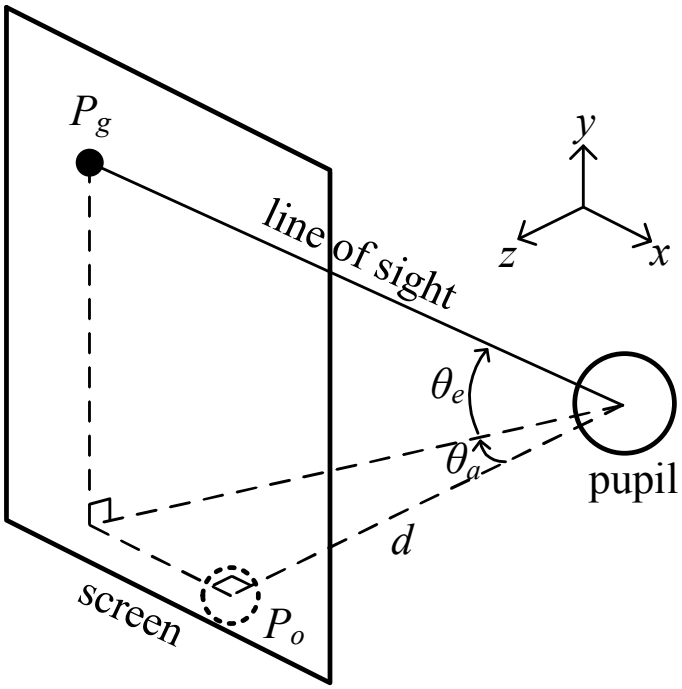


Figure 3.5: Rationale of the gaze point prediction.

3.2.2 Gaze Vector Measurement

The gaze vector is defined as the line of sight sourced from the user's pupil. For measuring the gaze vector of each eye, we use the iris landmarks, the eyeball center and iris center. Different from the gaze estimation in [29], which extracts the gaze vector from eyeball center and iris center, we follow a similar methodology in head pose measurement, that is solving the PNP problem from the 2D landmarks to their 3D locations. In particular, we compare the 2D location of these landmarks with a cardinal eyeball model, and solve a PNP problem to derive the rotation of the eyeball, including the yaw and pitch angles. Formally, denote the gaze vector from the left and right eye as $\boldsymbol{g_l}$ and $\boldsymbol{g_r}$ respectively. Assume that the yaw and pitch angle of the head pose are θ and ϕ, denote R_y and R_p as rotation matrices of yaw and pitch respectively, the transformed gaze vector $\boldsymbol{g_l^{'}}$ can be determined as follows:

$$\boldsymbol{g_l^{'}} = \boldsymbol{R_y R_p g_l} = \begin{bmatrix} \cos\theta\cos\phi & -\sin\theta & \cos\theta\sin\phi \\ \sin\theta\cos\phi & \cos\theta & \sin\theta\sin\phi \\ -\sin\phi & 0 & \cos\phi \end{bmatrix} \boldsymbol{g_l} \tag{3.3}$$

Similarly, the transformed gaze vector $\boldsymbol{g_r^{'}}$ from the right eye can be determined as the mirrored gaze vector of the left eye.

3.2.3 Gaze Point Measurement

With the 2D location of the landmark of iris, eyeball center and iris center detected, the gaze vector from the eye can be calculated following a model-based principle [29]. The theoretical rationale is illustrated in Figure 3.5. The gaze vector generally encodes the yaw θ_a and elevation θ_e of the line of sight, and the distance d along the normal from the user's iris center, to its projection point P_o on the surface where the screen is on. The relative translation between P_o and the gaze point P_g can be calculated in a model-based manner. Formally, take the location of the principle point $P_o(x_o, y_o)$, which is the center point of the screen, as the initial location of the user gaze point. The face-to-screen distance, denoted as d, is derivable from the transform vector on z-axis. Denote the Euler angle of the gaze vector on yaw as Θ_h, and similarly the Euler angle on pitch as Θ_e, the location of the gaze point $P_g(x_g, y_g)$ can be calculated as:

$$\begin{cases} x_g = x_o + d\tan\Theta_h \\ y_g = y_o + \frac{d\tan\Theta_e}{\cos\Theta_h} \end{cases} \tag{3.4}$$

When the user head rotates, the deviation in of the Euler angle $\Delta\theta_h$ and $\Delta\theta_e$ on yaw and pitch can be acquired from the rotation matrix respectively. The location of the gaze point $P_g(x_g, y_g)$ is then given by:

$$\begin{cases} x_g = x_o + d\tan(\Theta_h + \Delta\Theta_h) \\ y_g = y_o + \frac{d\tan(\Theta_e + \Delta\Theta_e)}{\cos(\Theta_h + \Delta\Theta_h)} \end{cases} \tag{3.5}$$

When the user is squinting at the screen, both the rotation and transform of the head impacts the movement of the gaze point. Denote the new face-to-screen distance as d', the yaw and pitch Euler angle deviation of the head as $\Delta\Phi_h$ and $\Delta\Phi_e$, the location of the gaze point $P_g(x_g, y_g)$ is then calculated as:

$$\begin{cases} x_g = x_o + d'\tan(\Theta_h + \Delta\Theta_h - \Delta\Phi_h) \\ y_g = y_o + \frac{d'\tan(\Theta_e + \Delta\Theta_e + \Delta\Phi_e)}{\cos(\Theta_h + \Delta\Theta_h - \Delta\Phi_h)} \end{cases} \tag{3.6}$$

3.3 Gaze-Based Text Entry

We use the gaze point measure as the location of a keystroke, as it is a virtual finger taping on the keys of a virtual keyboard. Another hint is required for indicting the happening of a keystroke. Acknowledging that the user's face expression can serve as such a hint, we consider it as an inconvenient option for the keystroke hint, decreasing the typing speed while increasing the cognitive load. As such, we use other potential input that a target system can receive. For the example of the desktop scenario, we use the left mouse button as the hint in our proof-of-concept prototype.

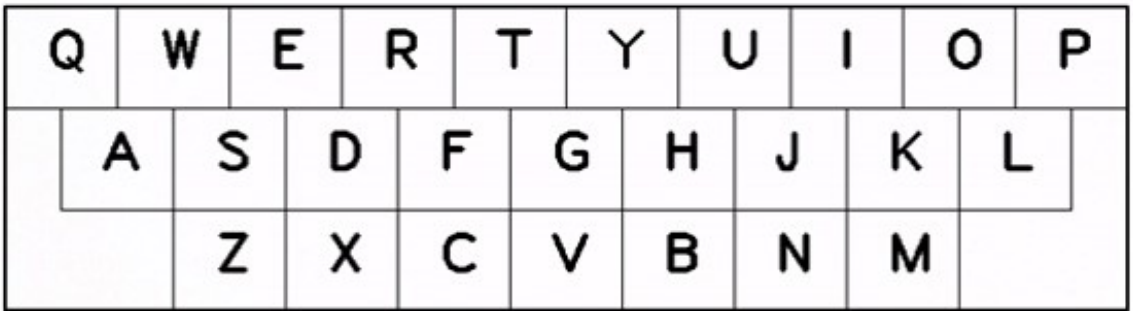


Figure 3.6: Layout of the virtual QWERTY keyboard.

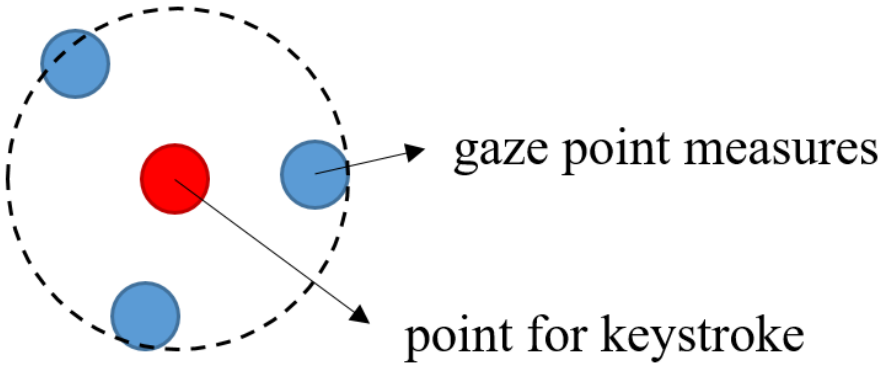


Figure 3.7: Keystroke point is taken from the recent three gaze point measures on the keyboard to stabilize the choice of keys.

3.3.1 Virtual Keyboard Layout

We develop a virtual QWERTY keyboard for the taping entry task, as shown in Figure 3.6. For the ease of implementation, the current keyboard consists of letters only, but it is considered as straightforward to extend the keyboard to other layout, such as the full QWERTY keyboard. When the user gaze point hovers over a virtual key, the system can detect which key is being focused based on the location of the gaze point. When a keystroke happens, the system will yield the chosen key and complete the entry of one letter.

3.3.2 Stabilizing Keystroke Location

As one challenge faced in the above method, the gaze point measure can be unstable along the time, due to the systematic source of error, such as the unstable results of face detection or face and eye landmark extractions. Additionally, the user can unintentionally move the head during the typing, causing unexpected movement of gaze points. To mitigate this challenge, we use the average of the gaze point location from the recent several frames, thus reducing the impact of deviation per frame. In our prototype implementation, we take

the average from the recent three frames, as shown in Figure 3.7. To further smooth the movement of gaze point on the keyboard, we interpolate the gaze point measure between two adjacent measures. As such, when the user gaze on the key, the system will try to make the focus as stable as possible, increasing the accuracy of the text entry.

Chapter 4

Experiments

In this chapter, we report the experiments on the proposed system, including the implementation of the proof-of-concept prototype and relative evaluation results.

4.1 Experiment Setup

As tackling the gaze-based text entry task that using common RGB cameras, we implement a proof-of-concept prototype that involves a desktop computer, a common RGB webcam, and a software implementation.

To quantitatively evaluate the performance of the key function modules of appearance-based feature extraction and model-based measurement, we mainly conduct in-dataset evaluations, i.e. comparing the derived result with the truth given in the datasets. For evaluations on the text entry performance with the prototype system, we test the real-world performance of one participant when conducting text entry task on the prototype system.

4.1.1 Hardware

We use a Logitech HD Webcam C525, a popular low-cost desktop webcam, as the RGB camera in our system. The system is implemented on a desktop, where an NVIDIA GeForce GTX 1080 Ti serves as the GPU for operating the DNN model.

4.1.2 Datasets

For eye feature extraction and gaze vector measurement, we use our synthesized eye image dataset, which provides detailed eye landmark locations and the gaze vector. We report more details of the dataset in Appendix A.

For head pose measurement, we use MPIIGaze dataset [47], a representative appearance-based dataset for gaze tracking tasks. The MPIIGaze dataset consists of over 200,000 webcam images collected from 15 participants in their daily desktop use, directly providing head pose measurement as baseline for benchmark.

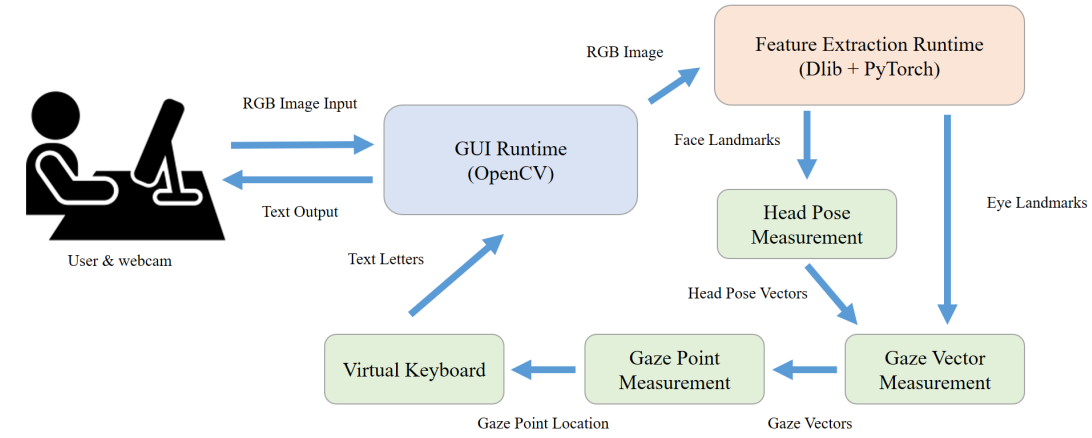


Figure 4.1: Framework and data flow of the prototype system. The blue part indicates the GUI, the orange part indicates the appearance-based feature extraction runtime, and the green parts are the implementations of model-based algorithms and the text entry function.

4.1.3 Implementation

The proof-of-concept prototype is implemented based on Python, a widespread cross-platform script programming language.

As illustrated in Figure 4.1, the prototype system mainly involves three open projects of libraries and platforms: OpenCV [3], Dlib [1] and PyTorch [4]. OpenCV is a popular library for image processing, and a well-founded base for realizing Graphic User Interface (GUI). In particular, we use the API from OpenCV to derive RGB image captured by the webcam, and utilize the UI module from OpenCV to provide graphic feedback of gaze point measurement, then text entry, on the screen. Dlib, as the toolbox for CV oriented machine learning algorithms, which is mainly used for face detection in the system. PyTorch serves as the DNN operation platform and is used to yield face and eye landmarks.

Beside of the operation of DNN models and realization of text entry tasks, the codes of the prototype mainly consists of logical implementation of model-based measurement algorithms in Section 3.2. Key codes that realize the model-based measurement and text entry task is listed in Appendix B.

Figure 4.2 shows a screenshot of the system, where the user is gazing at the key "W" and pressing the left mouse button to input the letter. The system is able to operate at a frame rate ranging from 15 to 22 frames per second (FPS), which is relatively limited due to imperfect implementations and hardware capacity, but feasible for evaluations.

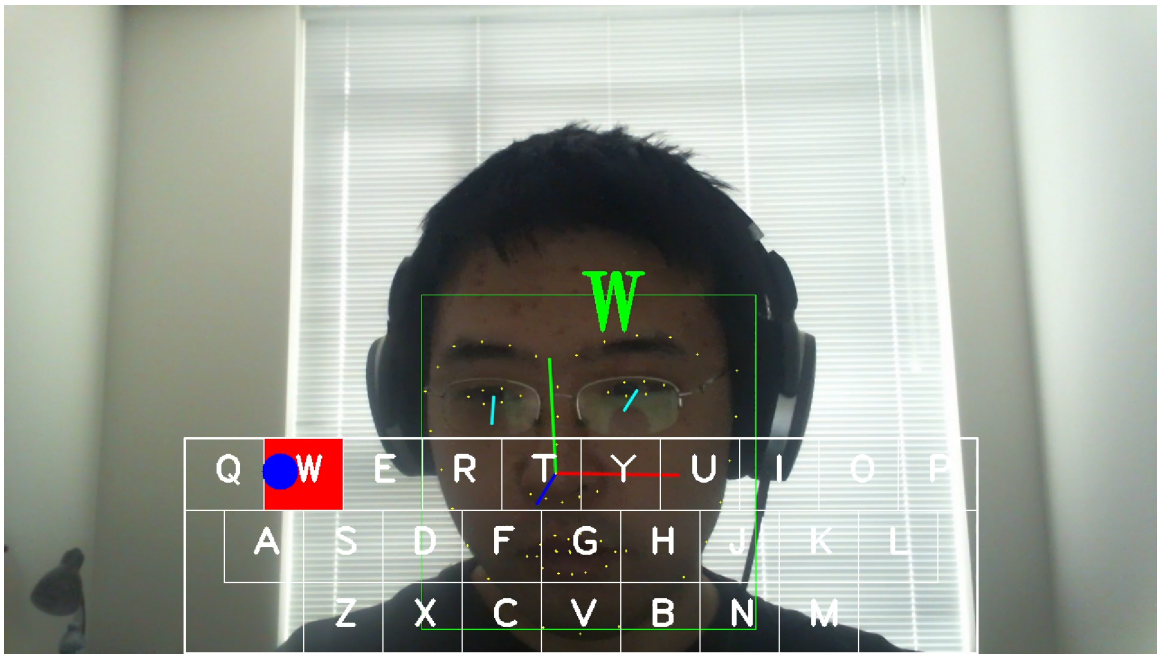


Figure 4.2: Screenshot of our prototype system when the user is conducting the text entry.

4.2 Evaluations

In this section, we report the quantitative evaluations on the performance of the appearance-based feature extraction and the model-based measurement, as well as the user performance in the gaze-based text entry task.

4.2.1 Results of Feature Extraction

Since we utilized pre-trained model for face feature extraction, we mainly focus on evaluating the accuracy of the eye feature extraction model.

Figure 4.3 illustrates the Cumulative Distribution Function (CDF) of the normalized deviation of the measured eye landmarks from the ground truth in the synthesized dataset. As shown in the evaluation results, the maximum normalized deviation is below 10% of the ground truth, and the overall mean accuracy is around 95%.

4.2.2 Results of Model-Based Measurement

The model-based measurement includes the gaze vector measurement and head pose measurement. For quantitative evaluation, we apply the pipeline of the system on our synthesized dataset and the MPIIGaze dataset, and derive the relative deviation of face landmarks, eye landmarks, head pose and gaze vector measurement from the ground truth.

The evaluation results on gaze vector measurement show that the maximum deviation of both yaw and pitch angles are close to around 5 degrees, as shown in Figure 4.4, and the mean deviation is over 4.5 degree for yaw angles below 4.5 degree for pitch angles.

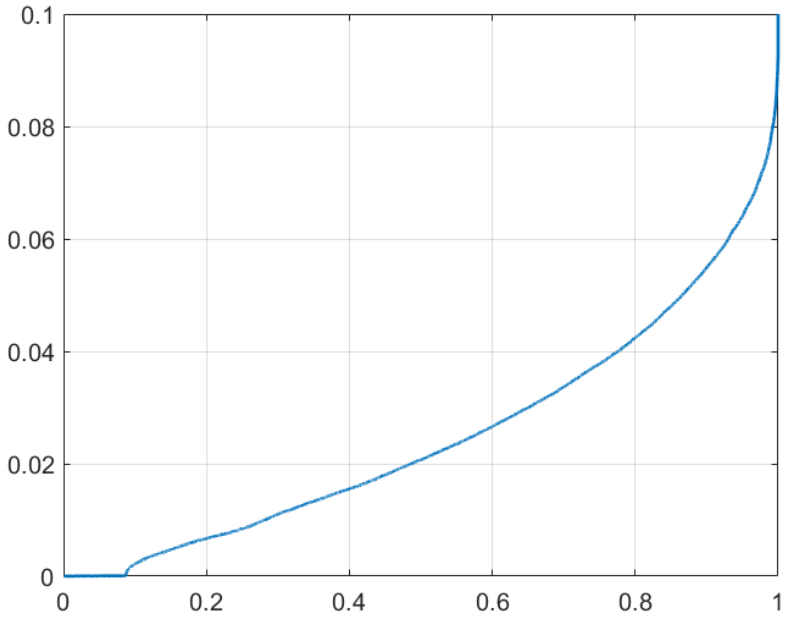


Figure 4.3: CDF of the normalized deviation of eye landmarks within the synthesized dataset.

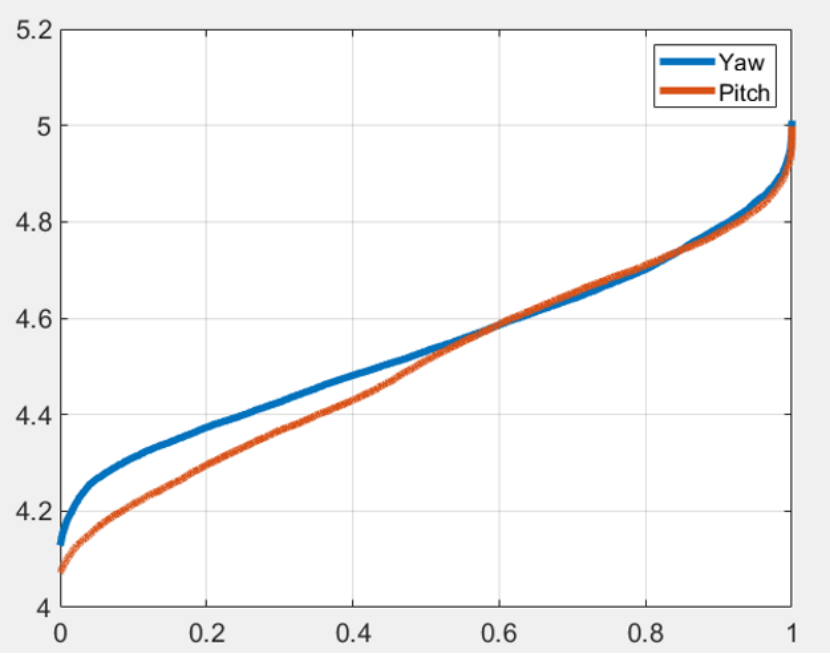


Figure 4.4: CDF of gaze vector angle deviation within the synthesized dataset.

For evaluating the head pose accuracy, we conduct measurement on all 15 participants' data in the MPIIGaze dataset and yield the mean deviations respectively.

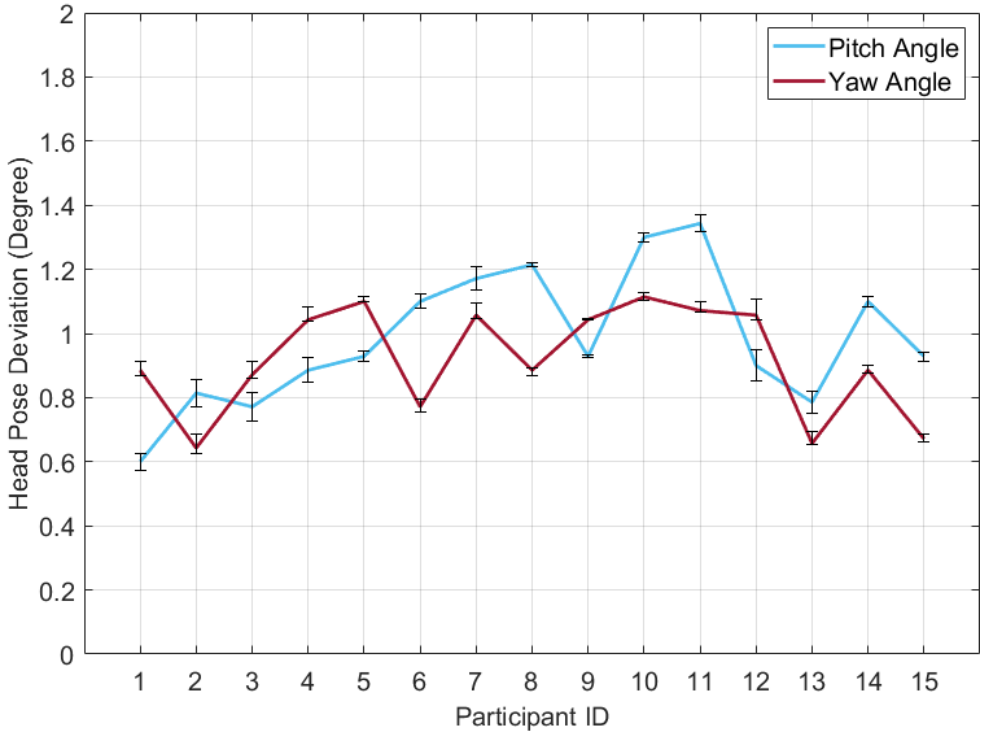


Figure 4.5: Angle deviation of both yaw and pitch angles in head pose measurement within the MPIIGaze dataset.

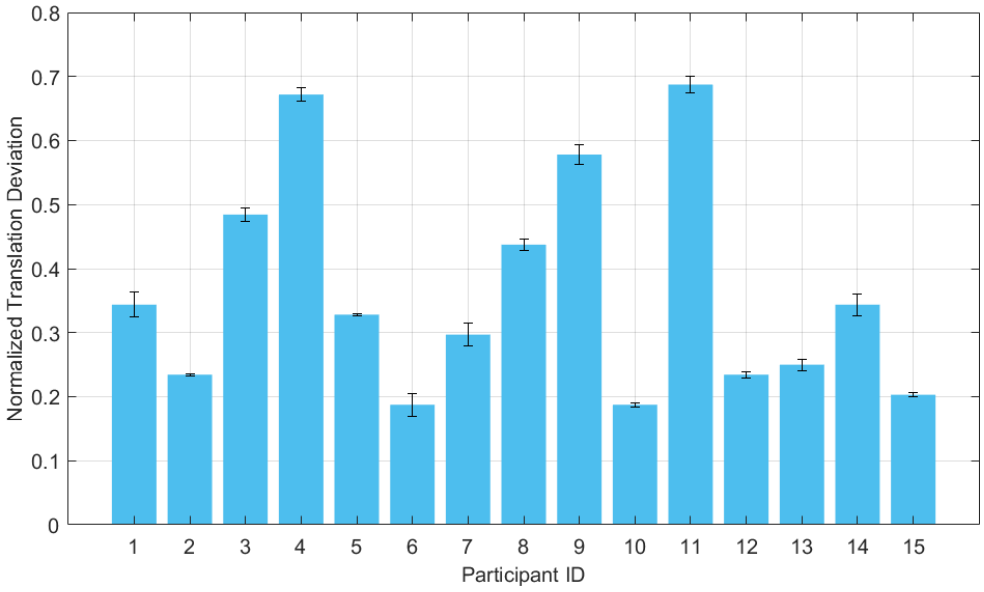


Figure 4.6: Translation deviation of head pose measurement within the MPIIGaze dataset.

As is shown in Figure 4.5, the overall angle deviation is within the range from 0.6 to 1.4 degree for both yaw and pitch angle measures, and the results of both types of angles are relatively close for each participant. Evaluation results on translation of head pose are illustrated in Figure 4.6. As shown in the figure, the overall deviation is below 0.7 for the normalized deviation of 3D distance between the translation measurement and the ground truth. One potential reason for the result is that we derive head pose measurement in the similar way with the derivation of the ground truth in the MPIIGaze dataset.

4.2.3 Results of Gaze-Based Text Entry

We conduct a transcribing task on the system to evaluate the text entry performance, including the speed and accuracy. We choose a letter that consists of 70 words and 342 letters from the Open American National Corpus [2] for the transcribing task. The detail of the chosen letter is presented in Appendix C.

The text entry task consists of 5 rounds with unlimited time, and 4 breaks of around 5 minutes. For each round we record the overall time to finish typing the letter by using the gaze, the count of the letters input, and the count of mistake letter input during the typing.

Table 4.1: Evaluation on the performance of gaze-based text entry

Task Round	Completion Time (min)	Overall Count	Mistake Count	Accuracy
1	17.8	374	32	91.4%
2	19.1	380	38	90.0%
3	18.6	380	38	90.0%
4	14.6	367	25	93.1%
5	11.3	370	28	92.4%
Avg.	16.3	374.2	32.2	91.4%

As shown in Table 4.1, the overall completion time for each round ranges from around 11 minutes to over 19 minutes. Therefore, the typing speed is measured as in a range of 18 to 30 letters per minute, and averagely 4.3 words per minute. There are totally 161 out of 1871 characters mischosen during the tasks, showing that the overall accuracy is over 90% and averagely around 91.4%

Chapter 5

Discussions and Conclusion

5.1 Discussions

The limitations of our solution are basically three-fold. First, our system is not quite complete as a text entry system, as the virtual keyboard consists of letters only. Additionally, the system does not provide much functionalities to ease the user's cognitive overhead, such as auto-fill and word suggestion. Second, the performance of our prototype is considerable low, mainly due to the fact that the implementation of the system is not optimized, and the hardware is a bit out-of-date for the latest open-source platforms. Third, the evaluation is conducted only in a preliminary manner, lacking extensive quantitative measurements on the user performance, while the sample space is also as small as possible.

As future works, we will first push the limitations above by completing and optimizing the system implementation. Furthermore, we will explore on leveraging existing advanced text entry techniques to improve the solution, notably including the popular functionalities like auto-fill and word suggestion, and other advanced text entry solutions, such as tap typing with optimized keyboard, swipe typing and phrase typing.

5.2 Conclusion

In this project, we propose a cost-efficient gaze-based text entry solution that uses common RGB cameras. We leverage the pros of appearance-based and model-based methodologies for gaze tracking, and tackle the challenges when use the gaze point measure as the hint for keystroke. From the evaluation on our proof-of-concept, we show that our system can enable a user to input texts by gazing at a virtual keyboard on the screen, at a speed ranging from 18 to 30 letters per minute with an accuracy of over 90%.

Bibliography

[1] Dlib c++ library, http://dlib.net/.

[2] Open american national corpus, https://anc.org/.

[3] Opencv python, https://pypi.org/project/opencv-python/.

[4] Pytorch, https://pytorch.org/.

[5] Seung-Jin Baek, Kang-A Choi, Chunfei Ma, Young-Hyun Kim, and Sung-Jea Ko. Eyeball model-based iris center localization for visible image-based eye-gaze tracking systems. *IEEE Transactions on Consumer Electronics*, 59(2):415–421, 2013.

[6] Syed Masum Billah, Yu-Jung Ko, Vikas Ashok, Xiaojun Bi, and IV Ramakrishnan. Accessible gesture typing for non-visual text entry on smartphones. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2019.

[7] Cunjian Chen. PyTorch Face Landmark: A fast and accurate facial landmark detector, 2021. Open-source software available at *https* : *//github.com/cunjian/pytorch*$_f$*ace*$_l$*andmark*.

[8] Wenzhe Cui, Rui Liu, Zhi Li, Yifan Wang, Andrew Wang, Xia Zhao, Sina Rashidian, Furqan Baig, IV Ramakrishnan, Fusheng Wang, et al. Glancewriter: Writing text by glancing over letters with gaze. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2023.

[9] Heiko Drewes, Alexander De Luca, and Albrecht Schmidt. Eye-gaze interaction for mobile phones. In *ACM Mobility*, 2007.

[10] Wenxin Feng, Jiangnan Zou, Andrew Kurauchi, Carlos H Morimoto, and Margrit Betke. Hgaze typing: Head-gesture assisted gaze typing. In *ACM Symposium on Eye Tracking Research and Applications*, pages 1–11, 2021.

[11] Jun Gong, Zheer Xu, Qifan Guo, Teddy Seyed, Xiang'Anthony' Chen, Xiaojun Bi, and Xing-Dong Yang. Wristext: One-handed text entry on smartwatch using wrist gestures. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2018.

[12] Yizheng Gu, Chun Yu, Zhipeng Li, Zhaoheng Li, Xiaoying Wei, and Yuanchun Shi. Qwertyring: Text entry on physical surfaces using a ring. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(4):1–29, 2020.

[13] Dan Witzner Hansen and Qiang Ji. 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, 2010.

[14] Zhenyi He, Christof Lutteroth, and Ken Perlin. Tapgazer: Text entry with finger tapping and gaze-directed word selection. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2022.

[15] Ramin Hedeshy, Chandan Kumar, Raphael Menges, and Steffen Staab. Hummer: Text entry by gaze and hum. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–11, 2021.

[16] Corey Holland, Atenas Garza, Elena Kurtova, Jose Cruz, and Oleg Komogortsev. Usability evaluation of eye tracking on an unmodified common tablet. In *ACM CHI Extended Abstracts on Human Factors in Computing Systems*, 2013.

[17] Michael Xuelin Huang, Jiajia Li, Grace Ngai, and Hong Va Leong. Screenglint: Practical, in-situ gaze estimation on smartphones. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2546–2557, 2017.

[18] Qiong Huang, Ashok Veeraraghavan, and Ashutosh Sabharwal. TabletGaze: unconstrained appearance-based gaze estimation in mobile tablets. *arXiv:1508.01244*, 2015.

[19] Qiong Huang, Ashok Veeraraghavan, and Ashutosh Sabharwal. TabletGaze: dataset and analysis for unconstrained appearance-based gaze estimation in mobile tablets. *Machine Vision and Applications*, 28(5-6):445–461, 2017.

[20] Sinh Huynh, Rajesh Krishna Balan, and JeongGil Ko. imon: Appearance-based gaze tracking system on mobile devices. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5(4):1–26, 2021.

[21] Pragma Kar, Krishna Mishra, Sudipro Ghosh, Sandip Chakraborty, and Samiran Chattopadhyay. Nosype: A novel nose-tip tracking-based text entry system for smartphone users with clinical disabilities for touch-based typing. In *Proceedings of the 23rd International Conference on Mobile Human-Computer Interaction*, pages 1–16, 2021.

[22] Kyle Krafka, Aditya Khosla, Petr Kellnhofer, Harini Kannan, Suchendra Bhandarkar, Wojciech Matusik, and Antonio Torralba. Eye tracking for everyone. In *IEEE CVPR*, 2016.

[23] Chandan Kumar, Ramin Hedeshy, I Scott MacKenzie, and Steffen Staab. Tagswipe: Touch assisted gaze swipe for text entry. In *Proceedings of the 2020 CHI conference on human factors in computing systems*, pages 1–12, 2020.

[24] Kuno Kurzhals, Fabian Göbel, Katrin Angerbauer, Michael Sedlmair, and Martin Raubal. A view on the viewer: Gaze-adaptive captions for videos. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2020.

[25] Zhenjiang Li, Mo Li, Prasant Mohapatra, Jinsong Han, and Shuaiyu Chen. iType: Using eye gaze to enhance typing privacy. In *IEEE INFOCOM*, 2017.

[26] Chen Liang, Chi Hsia, Chun Yu, Yukang Yan, Yuntao Wang, and Yuanchun Shi. Drg-keyboard: Enabling subtle gesture typing on the fingertip with dual imu rings. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 6(4):1–30, 2023.

[27] Zongjian Liu, Jieling He, Jianjiang Feng, and Jie Zhou. Printype: Text entry via fingerprint recognition. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 6(4):1–31, 2023.

[28] Yiqin Lu, Chun Yu, Shuyi Fan, Xiaojun Bi, and Yuanchun Shi. Typing on split keyboards with peripheral vision. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2019.

[29] Seonwook Park, Xucong Zhang, Andreas Bulling, and Otmar Hilliges. Learning to find eye region landmarks for remote gaze estimation in unconstrained settings. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research and Applications*, pages 1–10, 2018.

[30] Alexander Plopski, Yuta Itoh, Christian Nitschke, Kiyoshi Kiyokawa, Gudrun Klinker, and Haruo Takemura. Corneal-imaging calibration for optical see-through head-mounted displays. *IEEE transactions on visualization and computer graphics*, 21(4):481–490, 2015.

[31] Korok Sengupta, Sabin Bhattarai, Sayan Sarcar, I Scott MacKenzie, and Steffen Staab. Leveraging error correction in voice-based text entry by talk-and-gaze. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–11, 2020.

[32] Linfeng Shen, Yuchi Chen, and Jiangchuan Liu. Energy-efficient interactive 360° video streaming with real-time gaze tracking on mobile devices. In *2021 IEEE 18th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, pages 243–251. IEEE, 2021.

[33] Ludwig Sidenmark and Hans Gellersen. Eye&head: Synergetic eye and head movement for gaze pointing and selection. In *Proceedings of the 32nd annual ACM symposium on user interface software and technology*, pages 1161–1174, 2019.

[34] Shyamli Sindhwani, Christof Lutteroth, and Gerald Weber. Retype: Quick text editing with keyboard and gaze. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2019.

[35] Paul Streli, Jiaxi Jiang, Andreas Rene Fender, Manuel Meier, Hugo Romat, and Christian Holz. Taptype: Ten-finger text entry on everyday surfaces via bayesian inference. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2022.

[36] Kar-Han Tan, David J Kriegman, and Narendra Ahuja. Appearance-based eye gaze estimation. In *IEEE WACV*, 2002.

[37] Congyi Wang, Fuhao Shi, Shihong Xia, and Jinxiang Chai. Realtime 3D eye gaze animation using a single RGB camera. *ACM Transactions on Graphics (TOG)*, 35(4):118, 2016.

[38] Shibo Wang, Shusen Yang, Hailiang Li, Xiaodan Zhang, Chen Zhou, Chenren Xu, Feng Qian, Nanbin Wang, and Zongben Xu. Salientvr: saliency-driven mobile 360-degree video streaming with gaze information. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, pages 542–555, 2022.

[39] Erroll Wood, Tadas Baltrušaitis, Louis-Philippe Morency, Peter Robinson, and Andreas Bulling. Learning an appearance-based gaze estimator from one million synthesised images. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research and Applications*, pages 131–138, 2016.

[40] Erroll Wood and Andreas Bulling. Eyetab: Model-based gaze estimation on unmodified tablet computers. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, pages 207–210, 2014.

[41] Zheer Xu, Weihao Chen, Dongyang Zhao, Jiehui Luo, Te-Yen Wu, Jun Gong, Sicheng Yin, Jialun Zhai, and Xing-Dong Yang. Bitiptext: Bimanual eyes-free text entry on a fingertip keyboard. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2020.

[42] Zheer Xu, Yankang Meng, Xiaojun Bi, and Xing-Dong Yang. Phrase-gesture typing on smartphones. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, pages 1–11, 2022.

[43] Zheer Xu, Pui Chung Wong, Jun Gong, Te-Yen Wu, Aditya Shekhar Nittala, Xiaojun Bi, Jürgen Steimle, Hongbo Fu, Kening Zhu, and Xing-Dong Yang. Tiptext: Eyes-free text entry on a fingertip keyboard. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, pages 883–899, 2019.

[44] Huanpu Yin, Anfu Zhou, Guangyuan Su, Bo Chen, Liang Liu, and Huadong Ma. Learning to recognize handwriting input with acoustic features. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(2):1–26, 2020.

[45] Mingrui Ray Zhang, Shumin Zhai, and Jacob O Wobbrock. Typeanywhere: A qwerty-based text entry solution for ubiquitous computing. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2022.

[46] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. Appearance-based gaze estimation in the wild. In *IEEE CVPR*, 2015.

[47] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. Mpiigaze: Real-world dataset and deep appearance-based gaze estimation. *IEEE transactions on pattern analysis and machine intelligence*, 41(1):162–175, 2017.

Appendix A

Synthesized Eye Image Dataset

In this chapter, we report some more quantitative details about our synthesized eye image dataset.

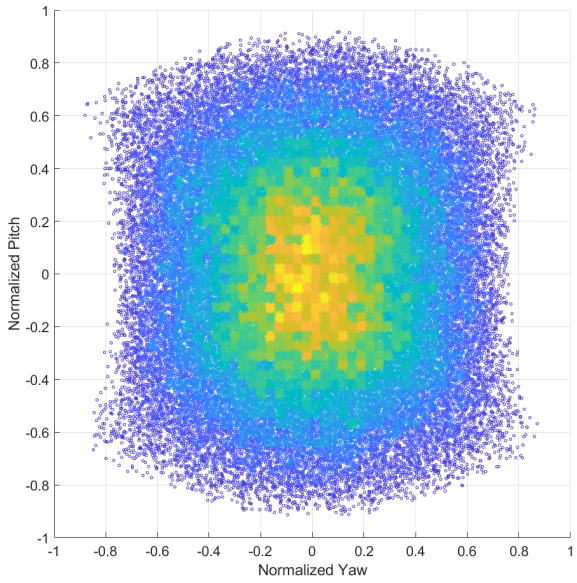
A.1 Head Pose and Gaze Vector Distribution

Our synthesized eye image dataset consist of 100,000 images that are generated from 3D eye region models. The synthesized dataset comes with parameters that include the 2D location of eye landmarks, the iris size that can be used to calculate the location of the eyeball center, and the Euler angle of the head pose against the camera.

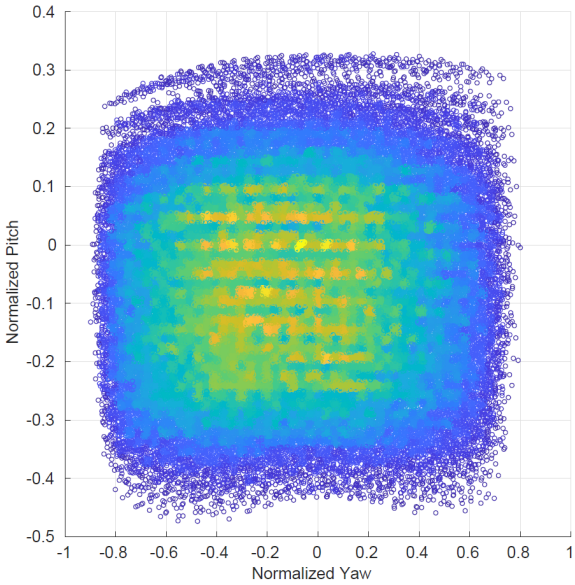
Evaluations on the distribution of the parameters, include the gaze vector and head pose, are illustrated in Figure A.1, where Figure A.1a shows the normalized yaw and pitch angle of the gaze vector and Figure A.1b shows the normalized angles of the head pose. The yaw and pitch angles of gaze vector are bounded within range $(-\frac{\pi}{2}, \frac{\pi}{2})$. The yaw angles of head pose are bounded within range $(-\frac{\pi}{4}, \frac{\pi}{4})$, while the pitch angles fall in range $(-\frac{\pi}{6}, \frac{\pi}{6})$.

A.2 Simulated Gaze Point Measurement

By setting the head location fixed, we simulate the gaze point tracking results using the synthesized dataset. The head is located at the normal from the center of the screen; as such, the gaze point measurement will be determined only by the distance of the head from the screen, the gaze vector and the head pose. Figure A.2 illustrates the normalized gaze point location on the screen as the simulation results. By setting the screen to be of a typical dimension of 60 cm by 40 cm, the simulated gaze points that fall out of the screen is filtered. As shown in the normalized results, the gaze point measurements can cover the screen and be uniformly distributed.



(a) Normalized yaw and pitch angles of gaze vector.



(b) Normalized yaw and pitch angles of head pose.

Figure A.1: Distributions of gaze vectors and head poses in the synthesized eye image dataset.

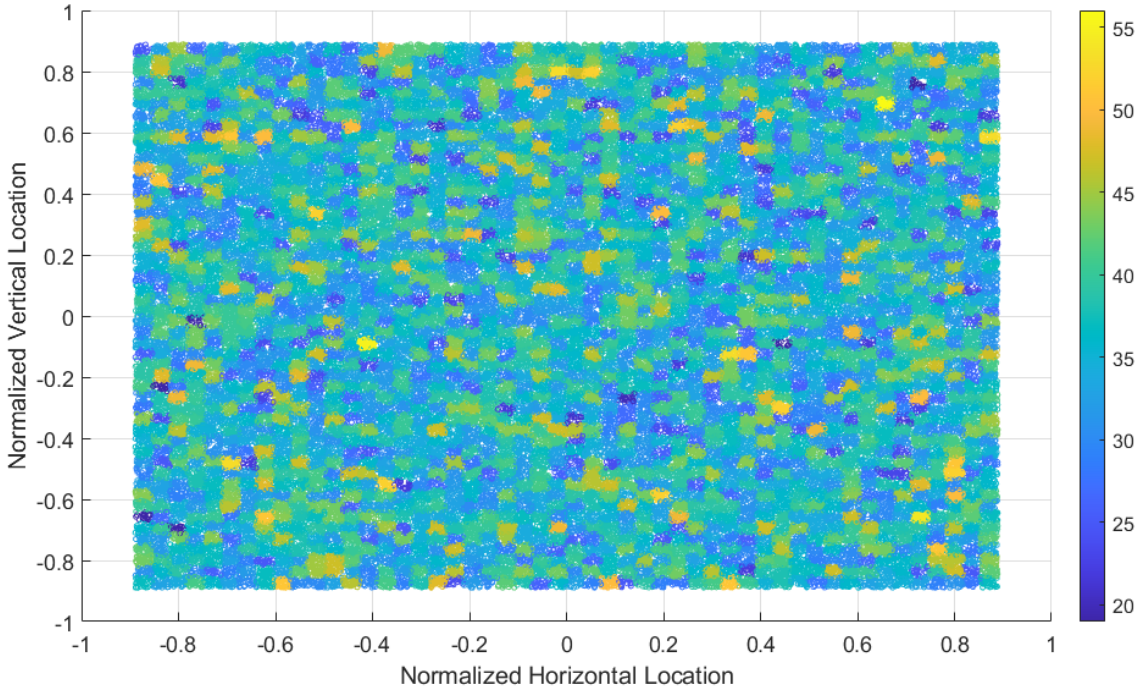


Figure A.2: Normalized simulation results of gaze point measurement on the screen within the synthesized dataset. The 3D location of the head is fixed to be on the normal emitted from the center of the screen. Gaze points that fall outside of the screen are filtered out.

Appendix B

Key Codes

In this chapter, we present the key codes that implement the algorithm parts in the project, including the codes for model-based measurement and the text entry task.

B.1 Codes for Model-Based Measurement

```
import cv2
import numpy as np
from scipy.spatial.transform import Rotation

# 68 landmarks on a cardinal face model
CARDINALFACE = np.array([
    [-0.07141807, -0.02827123, 0.08114384],
    [-0.07067417, -0.00961522, 0.08035654],
    [-0.06844646, 0.00895837, 0.08046731],
    [-0.06474301, 0.02708319, 0.08045689],
    [-0.05778475, 0.04384917, 0.07802191],
    [-0.04673809, 0.05812865, 0.07192291],
    [-0.03293922, 0.06962711, 0.06106274],
    [-0.01744018, 0.07850638, 0.04752971],
    [0., 0.08105961, 0.0425195],
    [0.01744018, 0.07850638, 0.04752971],
    [0.03293922, 0.06962711, 0.06106274],
    [0.04673809, 0.05812865, 0.07192291],
    [0.05778475, 0.04384917, 0.07802191],
    [0.06474301, 0.02708319, 0.08045689],
    [0.06844646, 0.00895837, 0.08046731],
    [0.07067417, -0.00961522, 0.08035654],
    [0.07141807, -0.02827123, 0.08114384],
    [-0.05977758, -0.0447858, 0.04562813],
    [-0.05055506, -0.05334294, 0.03834846],
    [-0.0375633, -0.05609241, 0.03158344],
```

```
[−0.02423648, −0.05463779, 0.02510117],
[−0.01168798, −0.04986641, 0.02050337],
[0.01168798, −0.04986641, 0.02050337],
[0.02423648, −0.05463779, 0.02510117],
[0.0375633, −0.05609241, 0.03158344],
[0.05055506, −0.05334294, 0.03834846],
[0.05977758, −0.0447858, 0.04562813],
[0., −0.03515768, 0.02038099],
[0., −0.02350421, 0.01366667],
[0., −0.01196914, 0.00658284],
[0., 0., 0.],
[−0.01479319, 0.00949072, 0.01708772],
[−0.00762319, 0.01179908, 0.01419133],
[0., 0.01381676, 0.01205559],
[0.00762319, 0.01179908, 0.01419133],
[0.01479319, 0.00949072, 0.01708772],
[−0.045, −0.032415, 0.03976718],
[−0.0370546, −0.0371723, 0.03579593],
[−0.0275166, −0.03714814, 0.03425518],
[−0.01919724, −0.03101962, 0.03359268],
[−0.02813814, −0.0294397, 0.03345652],
[−0.03763013, −0.02948442, 0.03497732],
[0.01919724, −0.03101962, 0.03359268],
[0.0275166, −0.03714814, 0.03425518],
[0.0370546, −0.0371723, 0.03579593],
[0.045, −0.032415, 0.03976718],
[0.03763013, −0.02948442, 0.03497732],
[0.02813814, −0.0294397, 0.03345652],
[−0.02847002, 0.03331642, 0.03667993],
[−0.01796181, 0.02843251, 0.02335485],
[−0.00742947, 0.0258057, 0.01630812],
[0., 0.0275555, 0.01538404],
[0.00742947, 0.0258057, 0.01630812],
[0.01796181, 0.02843251, 0.02335485],
[0.02847002, 0.03331642, 0.03667993],
[0.0183606, 0.0423393, 0.02523355],
[0.00808323, 0.04614537, 0.01820142],
[0., 0.04688623, 0.01716318],
[−0.00808323, 0.04614537, 0.01820142],
[−0.0183606, 0.0423393, 0.02523355],
[−0.02409981, 0.03367606, 0.03421466],
[−0.00756874, 0.03192644, 0.01851247],
[0., 0.03263345, 0.01732347],
[0.00756874, 0.03192644, 0.01851247],
[0.02409981, 0.03367606, 0.03421466],
[0.00771924, 0.03711846, 0.01940396],
[0., 0.03791103, 0.0180805],
```

```
    [-0.00771924, 0.03711846, 0.01940396],
    ], type=float)

# 10 landmarks on a cardinal eye model
CARDINALEYE = np.array([
    [0.0, 0.0, 0.0]
    [0.0, 0.0, -0.5],
    [0.0, 0.1, -0.4712415],
    [0.07071, 0.07071, -0.4712415],
    [0.1, 0.0, -0.4712415],
    [0.07071, -0.07071, -0.4712415],
    [0.0, -0.1, -0.4712415],
    [-0.07071, -0.07071, -0.4712415],
    [-0.1, 0.0, -0.4712415],
    [-0.07071, 0.07071, -0.4712415]], type=float)

# Derive head pose from 68 face landmarks by solving PNP
   problem
def getHeadPose(face_landmarks):
    # Camera parameters are necessary for solving PNP
    global camera_matrix, camera_coeff

    # Rotation vector on axis (x, y, z)
    rot_vec = np.zeros(3, dtype=float)

    # Translation vector on axis (x, y, z)
    trans_vec = np.array([0, 0, 1], dtype=float)

    # Solve PNP problem with cardinal face landmarks
    _, rot_vec, trans_vec = cv2.solvePnP(CARDINALFACE,
       face_landmarks, camera_matrix, camera_coeff, rot_vec,
        trans_vec, useExtrinsicGuess=True, flags=cv2.
       SOLVEPNP_ITERATIVE)

    # Rotation matrix from rotation vector
    rot_mat = Rotation.from_rotrans_vec(rot_vec)

    return rot_mat, trans_vec

# Derive gaze vector from 10 eye landmarks by solving PNP
   problem and adjust it with head pose
def getGazeVector(eye_landmarks, head_rot_mat,
   head_trans_vec):
    # Camera parameters are necessary for solving PNP
    global camera_matrix, camera_coeff

    # Rotation vector on axis (x, y, z)
```

```
    rot_vec = np.zeros(3, dtype=float)

    # Translation vector on axis (x, y, z)
    trans_vec = np.array([0, 0, 1], dtype=float)

    # Solve PNP problem with cardinal eye landmarks
    _, rot_vec, trans_vec = cv2.solvePnP(CARDINALEYE,
       eye_landmarks, camera_matrix, camera_coeff, rot_vec,
       trans_vec, useExtrinsicGuess=True, flags=cv2.
       SOLVEPNP_ITERATIVE)

    return rot_vec @ head_rot_mat, trans_vec +
       head_trans_vec

# Get 2D gaze point location
def getGazePoint(rot_vec, trans_vec):
    # take the screen surface as z == 0, as such the gaze
       point is the intersection point of gaze vector's
       extension
    # given rotation vector (rx, ry, rz) and translation
       vector (tx, ty, tz), the gaze point location is as:
    # x = tx + tz / rz * rx
    # y = ty + tz / rz * ry
    d = - trans_vec[2] / rot_vec[2]
    x = trans_vec[0] + d * rot_vec[0]
    y = trans_vec[1] + d * rot_vec[1]

    return (np.round(x), np.round(y))
```

B.2 Codes for Text Entry

```
import cv2
import numpy as np

# Function of adding text entry events to the frame
def showKeyboard(img):
    global gaze_point_loc, curText, keyboard_locs, hist_loc,
        counter, strike, reset
    # Take average of gaze point locations in last 3 frames
       as keystroke point
    if counter % 3 != 0 and not reset:
        hist_loc.append(gaze_point_loc)
    else:
        hist_loc = [gaze_point_loc]
```

```
avg_loc = np.mean(hist_loc, axis=0)

# draw keyboard on screen with resolution 1280 x 720
img = cv2.rectangle(img, (200, 480), (1080, 719), (255,
   255, 255), 2)
for row in range(3):
    for col in range(len(keyboard_locs[row])):
        if (keyboard_locs[row][col][0][0] <= avg_loc[0]
           <= keyboard_locs[row][col][1][0]) and
        (keyboard_locs[row][col][0][1] <= avg_loc[1] <=
           keyboard_locs[row][col][1][1]):
            if strike:
                color = (0, 0, 255)
            else:
                color = (255, 255, 0)
            img = cv2.rectangle(img, keyboard_locs[row][
               col][0], keyboard_locs[row][col][1],
               color, -1)
            break

# show text
if len(curText):
    img = cv2.putText(img, curText, (200, 440), cv2.
       FONT_HERSHEY_TRIPLEX, 2, (0, 0, 255), 2)

for row in range(3):
    if row == 0: # 10 keys of QWERTYUIOP
        for col in range(10):
            img = cv2.rectangle(img, keyboard_locs[row][
               col][0], keyboard_locs[row][col][1],
               (255, 255, 255), 1)
            img = cv2.putText(img, keyboard[row][col],
               keyboard_locs[row][col][2], cv2.
               FONT_HERSHEY_PLAIN, 3, (255, 255, 255),
               4)
    elif row == 1: # 9 keys of ASDFGHJKL
        for col in range(9):
            img = cv2.rectangle(img, keyboard_locs[row][
               col][0], keyboard_locs[row][col][1],
               (255, 255, 255), 1)
            img = cv2.putText(img, keyboard[row][col],
               keyboard_locs[row][col][2], cv2.
               FONT_HERSHEY_PLAIN, 3,
                                (255, 255, 255), 4)
    else: # 7 keys of ZXCVBNM
        for col in range(7):
```

```
                img = cv2.rectangle(img, keyboard_locs[row][
                    col][0], keyboard_locs[row][col][1],
                    (255, 255, 255), 1)
                img = cv2.putText(img, keyboard[row][col],
                    keyboard_locs[row][col][2], cv2.
                    FONT_HERSHEY_PLAIN, 3,
                                  (255, 255, 255), 4)

    return img

# Callback function, use left mouse button event as
   keystroke and add chosen letter to text
def callback(event, x, y, flags, param):
    global gaze_point_loc, strike, curText
    if event == cv2.EVENT_LBUTTONDOWN:
        strike = True
        for row in range(3):
            for col in range(len(keyboard_locs[row])):
                if (keyboard_locs[row][col][0][0] <=
                    gaze_point_loc[0] <= keyboard_locs[row][
                    col][1][0]) and (
                        keyboard_locs[row][col][0][1] <=
                            gaze_point_loc[1] <=
                            keyboard_locs[row][col][1][1]):
                    curText += keyboard[row][col]
                    break
    elif event == cv2.EVENT_LBUTTONUP:
        strike = False
```

Appendix C

Corpus for Text Entry Task

We choose the letter numbered 710CYL237 from the Open American National Corpus for conducting the evaluation on the text entry performance. The original content of the letter is as follows:

> Dear,
> Thank you for your interest in Aloha United Way. I am enclosing our 1999 Annual Report; Hoaloha, our quarterly report; and Year 2000 Report to Donors to introduce you to some basic services about AUW. If you need additional information, please feel free to contact me at (808) 543-2213, or e-mail me at: deborahm@auw.org.
> Sincerely,
> Deborah M. Sanders Murray Aloha United Way Assistant Vice President
> Marketing/ Communications Enc.

After clean the content by keeping only the letters and replacing the number and symbol with words, the text for conducting text entry task is as follows, including 70 words and 342 letters:

> Dear
> Thank you for your interest in Aloha United Way I am enclosing our Annual Report Hoaloha our quarterly report and Year Report to Donors to introduce you to some basic services about AUW If you need additional information please feel free to contact me at number or email me at deborahm at auw dot org
> Sincerely
> Deborah M Sanders Murray Aloha United Way Assistant Vice President
> Marketing Communications Enc