Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем

Development of an application to assist people with motor impairments in interacting with digital interfaces

Текстова частина до курсової роботи

за спеціальністю 121 «Інженерія програмного забезпечення»

Керівник курсової роботи:

Ст. викл. Кузьменко Д.О.

_____(підпис)

“____” _____ 2025 р.

Виконав студент БП-3 факультету інформатики:

Материнський В.В.

“____” _____ 2025 р.

Київ 2025 р.

Table of contents

Research Schedule

Number	Coursework writing stage	Dates
1	Preliminary acquaintance with the direction and the decision on the topic	2024-09-08 - 2024-09-22
2	Related work research	2024-10-01 - 2024-12-15
3	Experiment plan draft	2024-12-20 - 2025-02-23
4	Additional research and choice of technologies for implementation	2025-03-01 - 2025-03-05
5	Solution implementation, conclusions, and initial course paper draft.	2025-03-06 - 2025-03-20
6	Pre-final course paper draft	2025-04-20
7	Final course paper draft	2025-05-01

1. Introduction

Human-computer interaction (HCI) has evolved from being a convenience to becoming an essential component of modern life, enabling individuals to participate in social, professional, and scientific domains. However, conventional input devices such as keyboards, mice, and standard controllers present significant barriers to individuals with physical impairments, particularly those affecting the upper limbs.

In response to these challenges, a growing body of research has emerged over the past decades, driven by increasing awareness and commitment to inclusivity. Various assistive technologies have been developed, ranging from specialized controllers and voice-command systems to eye-tracking and gesture-based interfaces. These advancements reflect the dynamic nature of HCI research, which continuously seeks to accommodate the diverse needs of users.

The demand for accessible interaction technologies has intensified in recent years, spurred by global crises such as wars, natural disasters, and pandemics-each contributing to a rise in the population of individuals with disabilities. These developments underscore the urgent need for practical and inclusive interaction solutions.

Despite the availability of advanced assistive devices, their high cost often renders them inaccessible to the very users who need them most. This thesis aims to explore and propose affordable, user-friendly alternatives for human-computer interaction tailored to individuals with upper-limb impairments, with a focus on minimizing financial and technological barriers.

We conducted a study of current gaze-tracking solutions, including those requiring specialized hardware and those functioning with a standard camera. Based on this analysis, we developed a lightweight JavaScript class that implements gaze-based navigation using the **GazeCloudAPI** and **WebGazer.js**

libraries. It employs a virtual cursor to simulate mouse actions and supports flexible configuration, including optional click-free calibration and adjustable interaction timing. Written in native JavaScript, the class is framework-agnostic and can be seamlessly integrated into any modern web application.

1.1 Preliminaries

HCI or human-computer interaction is a field of study which focuses on design and developing interfaces which allow people to interact with computers.[1]

HCI studies focus on user psychology and attempt to make human-computer interaction like human-to-human interaction.

According to INS (International Neuromodulation Society) [2] motor impairment is the partial or total loss of function of a body part, usually a limb or limbs. This can be caused by different diseases such as Parkinson's or sclerosis.

WHO (World Health Organisation) claim that physical disability can be defined as limitations in performing activities at certain standards of quality and efficiency and as restrictions in participating in family and social life [3].

European disability forum says that starting from 2022, nearly 300.000 people in Ukraine sustained injuries that caused some type of disability.

Assistive technologies that create interface for HCI are divided into two different categories: hardware- and software-based.

Hardware based solutions are different switches, mice, keyboards and other devices that are adapted to special needs of people with impairments. For example, Fig.1 showcases a set of Microsoft accessories that were designed to replace traditional electronic mice for people with disabilities.

Fig.1 Hardware for eye-tracking

Software-based solutions are voice assistants like Siri or Alexa - they can allow individuals with disabilities to accomplish their everyday tasks through voice commands. Screen readers which convert text to speech or braille are an example of such technology - they are crucial for people with vision impairments.

Both software- and hardware-based technologies play an important role in making access to computer easier, though both have their drawbacks and limitations. Hardware-based technologies are often expensive and not produced in high volumes. Software-based solutions are more affordable and available, yet they tend to be less precise especially in domain-specific applications.

We chose to explore vision-based control, specifically gaze tracking, as we think Natural Language Processing (NLP) and speech-related technologies received a lot of attention in past five years in scientific and commercial application. There has been a rise of Large Language models (LLMs), virtual assistants, smart speakers and other availability tools.

On the other hand, vision-based interaction remains less covered but still very promising area for exploration. It offers silent, hand-free control which can be more convenient for people who are not willing to attract attention, or in spaces where silence is desirable (e.g. libraries). Gaze-based controls are a natural way of controlling the interface, because when the pointer is moved or clicked, we always concentrate our gaze on it rather than verbalize our intent to click on some element.

This research focuses on gaze-based interaction solutions designed to support individuals with physical or motor impairments. Over the past several decades, this topic has garnered significant attention in both academic and commercial domains. The present study aims to contribute to this body of work by proposing an integrative approach that enhances web accessibility for users with special needs.

2. Related works

Research on eye-tracking for human computer interaction started in early 1980s. Richard Bolt was one of the pioneers with “Eyes at the interface” [4] and “Gaze-orchestrated dynamic windows” [5] works.

In his work, Bolt proposed that eye-tracking technologies could enhance human-computer interaction, particularly in supporting multitasking and shifting attention between tasks. He emphasized that, at the time, most research focused on behavioural insights rather than on using gaze as a direct input for system control. Interestingly, this trend persists today. Much of the current eye-tracking research still centres on market analysis and user insights, rather than exploring gaze as an interactive input method for communication with computer systems.

Early eye-tracking research was dominated by solutions that relied on additional hardware. These systems typically required an infrared detector positioned below or above the display, using corneal reflection to determine the user's gaze point on the screen. At that stage, the focus was on capturing actions such as blinking or eye movements. The core of the process involved calculating screen coordinates based on recorded eye data. Due to technological constraints, most of these computations were handled through specialized hardware setups rather than software.[6]

For example, Robert Jacob’s “Eye Movement-Based Human-Computer Interaction” [7] used Applied Science Laboratories eye tracker for their experiments. The limitation of such solutions is that they need the user to be in strict position. For instance, have their head positioned strictly at the distance of 60cm away from the screen.

Recent years have seen a significant rise in the use of machine learning for gaze estimation. For instance, Mahender K. et al.[8] developed a client-server gaze-tracking application in which the backend utilized OpenCV along with Support

Vector Machines and Neural Networks to predict the user's gaze point on the screen. Their system demonstrated that combining traditional computer vision techniques with machine learning can achieve accurate and responsive gaze tracking.

There were also studies that incorporated machine learning algorithms, though often abstracted behind user-friendly interfaces. One such example is **WebGazer.js** [9], which aimed to make gaze prediction models more accessible through a simple JavaScript interface. It offers a degree of customization and operates without the need for complex calibration. Our work builds on this approach by leveraging **WebGazer's** functionality alongside another open-source tool - GazeRecorder - and its associated **GazeCloudAPI**[10].

3. Methods

Native JavaScript was utilized to render the virtual cursor and handle data processing. For gaze-tracking functionality, two libraries - **GazeCloudAPI** and **WebGazer.js** - were employed.

All application logic was encapsulated within a JavaScript class, facilitating seamless integration with any web page or front-end framework.

Gaze prediction data obtained from the models provided by **WebGazer.js** and **GazeCloudAPI** was processed through their respective interfaces to enable vision-based navigation. At the foundational level, each set of coordinates received from either library was temporarily stored in a buffer. Once the buffer reached a size of five entries, the mean values of the *x* and *y* coordinates were computed. These averaged coordinates were then used to trigger a move event for both the virtual and logical cursors.

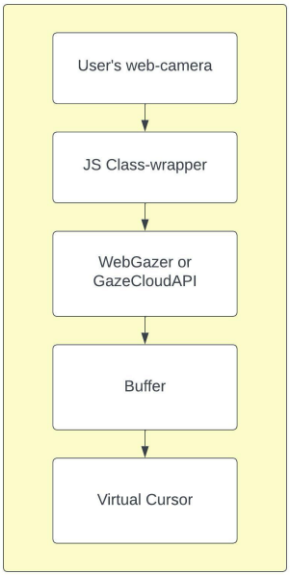

Fig. 2 Data pipeline for eye-tracking cursor

When the cursor entered an interactive element, such as a link or a button, a timeout is initiated. If the cursor remains within the bounds of that element for

at least 500 milliseconds (a configurable threshold), a click event is triggered, enabling interaction with the element.

Additionally, when the cursor enters the top or bottom 10% of the screen, scroll events were programmatically generated. These events allow for an upward or downward navigation through the page using only gaze input.

4. Experiments

Objective of experiments

The objective of our experiments is to develop a lightweight application capable of seamless integration with the frontend layer of any modern web application. The solution is intended to be easy to configure and adapt across various platforms.

4.1 Scope of experiments

Initially, we considered developing APIs that would allow integration of existing eye-tracking models implemented using Python libraries such as **OpenCV**. However, we decided to leave these approaches for future work due to their complexity and computational demands. Consequently, we narrowed the focus to building a lightweight wrapper around existing web-based APIs - specifically, **WebGazer.js** and **GazeCloudAPI** - to create an interaction interface for users with motor impairments.

4.2 Initial considerations

Early in the development process, we explored Python-based solutions under the assumption that the application would consist of both frontend and backend components. One of the models tested was **GazeTracking** [11], which is based on OpenCV and provides an API suitable for backend integration.

Upon deploying the model on a local machine, we encountered several limitations:

- A significant volume of data had to be constantly exchanged between the frontend and backend components.
- The computational load required by the model rendered it unsuitable for real-time applications or lightweight deployments.

As a result, we shifted our focus to frontend-based solutions that offer preconfigured APIs. This allowed us to concentrate on enabling user interaction with the web interface, rather than investing resources in bridging backend models with frontend functionality.

4.3 Wrapper

For the target application, we selected **WebGazer.js** - a web-camera-based eye-tracking library designed specifically for browser environments. During the development of the demonstration website and the gaze-controlled interaction layer, we implemented user interface controls using **React** components and its associated functionality [12].

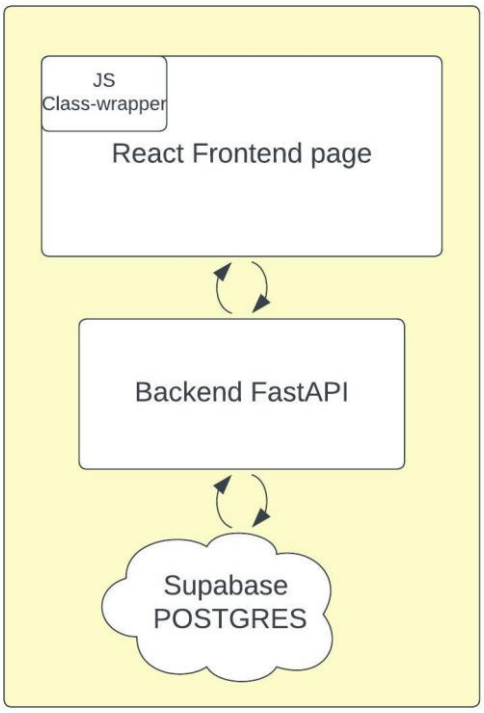

Fig. 3 Demo-website high-level architecture

At this stage, the application was tightly coupled with the demo web page. This coupling was identified as a limitation, as the primary objective of this research

is to develop a reusable, modular solution that can be easily integrated into any front-end framework.

To resolve this issue, we refactored the codebase by extracting all logic related to eye-tracking and user interaction into a dedicated JavaScript class. This modularization enabled the decoupling of core application logic from specific UI implementations. As a result, the solution became framework-agnostic, allowing for straightforward integration with a wide range of web frameworks and interfaces.

4.4 Interaction logic

The initial strategy for processing gaze data from the **WebGazer.js** library was developed based on an intuitive observation: during typical computer use, users tend to follow the cursor with their gaze. In the earliest version of our implementation, each new gaze prediction directly triggered a mouse event, thereby mapping gaze movement to cursor interaction in real time.

Due to JavaScript's inherent limitations in manipulating the system-level cursor, all interaction logic was implemented through a **virtual cursor** rendered within the webpage. With each new gaze prediction, the virtual cursor was programmatically moved to the predicted screen coordinates.

To simulate click events, the system continuously monitors the position of the virtual cursor relative to interactive page elements. Upon entering such an area, a timeout mechanism is activated. If the cursor remains within the bounds of the target element for a predefined duration - without exiting the area - a synthetic mouse click event is dispatched. This mechanism served as a proxy for intentional user interaction based on sustained gaze fixation.

Furthermore, scrolling functionality is incorporated by defining threshold regions at the top and bottom 10% of the viewport height. When the virtual

cursor entered these regions, the system generated corresponding scroll events- upward for the top zone and downward for the bottom zone. This feature both enhanced navigational capabilities for the user and contributed to improved model performance, as gaze predictions were typically more stable and accurate within the central region of the screen.

4.5 Challenges

Customization

The primary limitation of our approach lies in the reliance on a **third-party API** rather than an open-source model. This choice restricted our ability to customize and fine-tune the system, which also had an impact on overall accuracy. We have identified this as an area for future work, with plans to explore more flexible, open-source alternatives in subsequent iterations of the project.

Erratic Cursor Movement

One of the initial challenges encountered with this approach was the **erratic movement** of the virtual cursor. The gaze-tracking model was unable to consistently predict identical coordinates, even when the user's gaze remained fixed on a single point. This variability is primarily due to environmental factors such as slight head movements and lighting conditions, which affect prediction stability. As a result, the virtual cursor tended to oscillate around the intended focal point, which created a visually distracting experience and diminished the overall usability of the system.

To mitigate this issue, a simple **heuristic smoothing technique** was introduced. A buffer mechanism was implemented to accumulate a series of predicted gaze points, after which the system calculated their average coordinates. Through empirical testing, a buffer size of five predictions was found to provide a satisfactory balance between responsiveness and stability. This adjustment

significantly improved the smoothness of cursor movement, resulting in a more user-friendly and visually comfortable experience.

Prediction Accuracy

A second major limitation was the **accuracy** of gaze prediction. One of the key advantages of the **WebGazer.js** library is that it operates without requiring user calibration, thereby simplifying setup and enhancing convenience. However, the absence of a calibration phase led to reduced prediction precision and inconsistency in tracking accuracy.

To address this shortcoming, we investigated alternative gaze-tracking solutions that incorporate a calibration mechanism. After evaluating several candidates, we selected **GazeCloudAPI**, which includes built-in calibration functionality and demonstrated higher accuracy during testing.

Although the use of **GazeCloudAPI** improved prediction reliability, it still did not meet the accuracy required for interacting with smaller user interface elements. As a result, we explored a **vector-based approach** that aimed to guide cursor movement based on directional changes rather than absolute gaze coordinates. However, this method introduced non-intuitive cursor behavior, ultimately degrading the overall user experience.

Based on these findings, we opted to return to our original implementation hypothesis, prioritizing simplicity and visual consistency over complex directional models.

Results

We developed a JavaScript class that encapsulates the core control logic required for navigating web pages using two powerful eye-tracking APIs: **GazeCloudAPI** and **WebGazer.js**. The solution is lightweight and compatible with most modern computing devices. Additionally, the implementation allows for developer-level customization, such as configuring the buffer size and interaction delay.

For demonstration purposes, we devised a lightweight three-tier application architecture. The frontend is implemented using the **React** framework, the backend is built with **Python's FastAPI**, and the database layer utilizes **PostgreSQL**, hosted via **Supabase**.

The frontend consists of several navigable pages designed to showcase the eye-tracking functionality. It communicates with the backend through a dedicated feedback page only, which includes a simple form equipped with radio buttons. This design allows users to complete the form entirely through gaze-based interaction.

The backend exposes a single endpoint that receives user feedback from the frontend and stores the submitted data in the database.

Given the use of two different APIs, we conducted a comparative performance analysis, summarized in the table below:

Model	Inference time, ms	CPU, %	RAM, Mb
GazeCloudAPI	37	10%	31
WebGazer.js	45	14%	42

Table 1. Performance comparison between **GazeCloudAPI** and **WebGazer.js**.

The results indicate that **GazeCloudAPI** outperforms **WebGazer.js** both in execution speed and overall resource efficiency. It is important to note, however, that CPU usage figures are approximate, as JavaScript does not natively provide precise tools for measuring processor load on the frontend. All measurements were conducted on a 2021 MacBook Pro with an Apple M1 Pro chip and 32 GB of RAM.

Based on our evaluation, the most effective solution at this stage is to use our application with **GazeCloudAPI** as the underlying eye-tracking engine. Although its calibration process is less convenient, it requires minimal user interaction and significantly enhances tracking accuracy, making it a reasonable trade-off in this context. It delivers a smoother and more reliable user experience.

In contrast, **WebGazer.js** does not require explicit calibration, which makes it easier to set up. However, this convenience comes at the cost of lower prediction accuracy. While users may attempt to improve its performance by interacting with the screen (e.g., clicking to aid calibration), this compromises its "no-calibration" advantage and ultimately makes it a less favorable option.

Despite **GazeCloudAPI's** superior performance overall, it still exhibits limitations in terms of fine-grained precision. While interaction with small interface elements is possible, it remains difficult and, in some cases, nearly unfeasible for users requiring high-precision control.

Conclusion and future prospects

Using plain JavaScript and two different gaze-tracking libraries, we developed a lightweight plug-in class that enables users to navigate web pages using only their gaze. The tool is easy to configure for developers and intuitive to use for end-users. It is also framework-agnostic, making it compatible with any JavaScript-based website. With minimal resource consumption and no external dependencies, it offers an accessible solution for integrating gaze-tracking interaction into modern web applications.

Despite its efficiency and ease of use, the tool has several notable limitations. To achieve maximum precision, it requires calibration each time it is used and bright light conditions. Even after calibration, accuracy remains insufficient for interacting with smaller interface elements. Performance also degrades significantly in low-light conditions or when using lower-resolution cameras. The underlying models in the APIs are relatively outdated and lack flexibility for further training or parameter tuning. Additionally, these models cannot leverage GPU acceleration and are limited to CPU execution. Finally, the JavaScript class currently does not support interactions with text fields and is limited to basic elements like buttons and switches.

To further improve this work, a valuable next step would be to train a custom gaze-tracking model or adopt a more modern pre-trained model and integrate it via a dedicated wrapper. This would provide greater flexibility for fine-tuning and optimizing performance for specific use cases. Additionally, implementing a virtual keyboard as part of the JavaScript class would significantly expand interaction capabilities by enabling text input. A slight redesign of the navigation logic could also enhance usability - shifting from the current "point and focus" mechanism to a "point and interact" approach could provide a more intuitive and responsive user experience.

Acknowledgements

We would like to acknowledge that the formatting of this work was carried out with the assistance of ChatGPT, a large language model developed by OpenAI. We confirm that its use was strictly limited to formatting and layout tasks; all ideas, analyses, and written content presented here were created solely by the authors.

Citations

1. Interaction Design Foundation - IxDF. (2016, June 6). What is Human-Computer Interaction (HCI)? Interaction Design Foundation - IxDF. https://www.interaction-design.org/literature/topics/human-computer-interaction

2. "Motor Impairment." International Neuromodulation Society, www.neuromodulation.com/motor-impairment. Accessed 21 Apr. 2025.

3. Miyahara, M., Piek, J., & Rigoli, D. (2024). Physical disabilities. In W. Troop-Gordon & E. W. Neblett (Eds.), Encyclopedia of Adolescence (2nd ed., pp. 404–416). Academic Press. https://doi.org/10.1016/B978-0-323-96023-6.00045-2

4. Bolt, R. A. (1982). Eyes at the interface. In Proceedings of the 1982 Conference on Human Factors in Computing Systems (CHI '82) (pp. 360–362). Association for Computing Machinery. https://doi.org/10.1145/800049.801811

5. Richard A. Bolt. 1981. Gaze-orchestrated dynamic windows. SIGGRAPH Comput. Graph. 15, 3 (August 1981), 109–119. https://doi.org/10.1145/965161.806796

6. Sharma, Anjana & Abrol, Pawanesh. (2013). Eye Gaze Techniques for Human Computer Interaction: A Research Survey. International Journal of Computer Applications. 71. 18-25. 10.5120/12386-8738.

7. Jacob, Robert. (2003). Eye Movement-Based Human-Computer Interaction.

8. K, Mahender & A, Mahesh & A, Malika & E, Mallesh & Neelam, Malleswari & O, Mallikarjun & Karthik, Ragipati. (2023). Eye-Tracker to Control Pointer Using Machine Learning. International Journal for Research in Applied Science and Engineering Technology. 11. 627-630. 10.22214/ijraset.2023.57416.

9. Papoutsaki, Alexandra & Sangkloy, Patsorn & Laskey, James & Daskalova, Nediyana & Huang, Jeff & Hays, James. (2016). WebGazer: Scalable Webcam Eye Tracking Using User Interactions.

10. GazeRecorder. (n.d.). GazeRecorder and GazeCloudAPI. Retrieved April 21, 2025, from https://www.gazerecorder.com/

11. Lamé, A. (2019). *GazeTracking* [Computer software]. GitHub. https://github.com/antoinelame/GazeTracking

12. Meta. *React: A JavaScript Library for Building User Interfaces*. 2024. https://reactjs.org/