Chromium based Framework to Include Gaze Interaction in Web Browser

Chandan Kumar[1]
kumar@uni-koblenz.de

Raphael Menges[1]
raphaelmenges@uni-koblenz.de

Daniel Müller[1]
muellerd@uni-koblenz.de

Steffen Staab[1,2]
staab@uni-koblenz.de

[1]Institute for Web Science and Technologies (WeST)
University of Koblenz-Landau, Germany

[2]Web and Internet Science Research Group (WAIS)
University of Southampton, UK

ABSTRACT

Enabling Web interaction by non-conventional input sources like eyes has great potential to enhance Web accessibility. In this paper, we present a Chromium based inclusive framework to adapt eye gaze events in Web interfaces. The framework provides more utility and control to develop a full-featured interactive browser, compared to the related approaches of gaze-based mouse and keyboard emulation or browser extensions. We demonstrate the framework through a sophisticated gaze driven Web browser, which effectively supports all browsing operations like search, navigation, bookmarks, and tab management.

Keywords

Web browser; Chromium Embedded Framework; DOM node extraction; interactive elements; webpage rendering; gaze input; eye-controlled interfaces

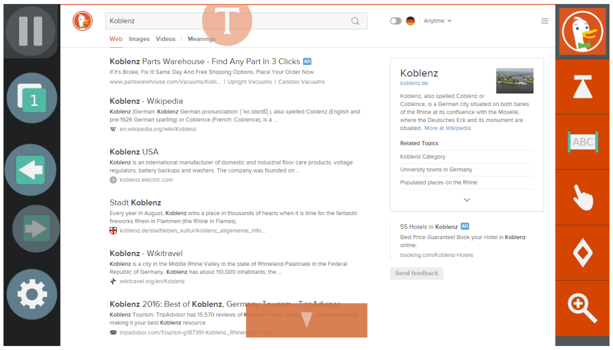
1. INTRODUCTION

The Web browser enables the end-user to perform various tasks with online technologies such as reading news, interacting in social networks, watching videos, editing photos or even working with an office suite. Although the tasks are different, the underlying data structure on the client side is built with HTML, CSS, JavaScript, and the user interactions are designed for mouse and keyboard or touch input. Therefore, the Web browser can accomplish modern computer tasks in analogous manner, since the Web applications' user interface is written in the same languages and designed to interact with the same set of input devices. However, a novel input mechanism such as eye tracking does not assimilate inherently in the current Web browsers.

Eye tracking as an input source and assistive technology [4] is useful to build hands-free Web interfaces, especially for cases where motor impairments hinder easy hand and body motion. Web access and browser control by gaze interaction can be achieved by two methods: either the eye tracker is simply used to control the mouse and keyboard in the normal graphical user interface, or the second approach of browser extensions/customized interfaces to integrate eye gaze signals in the browser. The performance of pure emulation of mouse input on operating system level [1, 2] highly depends on the eye tracking accuracy, and suffers with low usability in the sophisticated scenario of Web browsing. To overcome this issue, there have been some recent approaches to develop browser extensions and prototypes for Web browsers [11]. In addition to acclimatize the interaction design, the major challenge for these approaches is to identify the interactive Web elements and to include eye gaze events with the Web technology.

In this regard, the Text 2.0 framework [3] had introduced a novel benchmark to mix eye tracking data with Web technology, where gaze-responsive webpages can be implemented via interpreting a new set of gaze handlers (e.g., `onFixation`, `onGazeOver` and `onRead`) that can be attached to parts of the DOM tree and behave similar to existing HTML and JavaScript mouse and keyboard event facilities. Wassermann et al. [12] built upon this concept to enable eye gaze events in eLearning environment. Although, it is a pertinent guideline for the Web developers to include eye gaze interactions in their application, it does not resolve the problem of browsing the current Web with eye-based interactions. Hence there is a need of Web extraction methodology to identify the input and selectable objects, and a gaze enhanced user centered design to interact with the objects. There has been some elementary approach in this direction to identify basic elements such as overflows and hyperlinks [11, 10]. However Web is much more complicated and requires a comprehensive and scalable framework to identify the interactive elements, and to design suitable interaction for all browsing functionalities.

WWW'17 Companion, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4914-7/17/04.
http://dx.doi.org/10.1145/3041021.3054730



Koblenz

Web Images Videos | Meanings

Anytime

Koblenz Parts Warehouse - Find Any Part In 3 Clicks AD

If It's Broke, Fix It! Same Day And Free Shipping Options. Place Your Order Now.

www.partswarehouse.com/Vacuums/Kobl... | Upright Vacuums | Canister Vacuums

Koblenz - Wikipedia

Koblenz (German: **Koblenz** German pronunciation: [ˈkoːblɛnt͡s]), also spelled Coblenz (English and pre-1926 German spelling) or Coblence (French: Coblence), is a ...

en.wikipedia.org/wiki/Koblenz

Koblenz USA

Koblenz is an international manufacturer of domestic and industrial floor care products, voltage regulators, battery backups and washers. The company was founded on ...

koblenz-electric.com

Stadt **Koblenz**

Every year in August, **Koblenz** wins a place in thousands of hearts when it is time for the fantastic fireworks Rhein in Flammen (the Rhine in Flames).

koblenz.de/stadtleben_kultur/koblenz_allgemeine_info...

Koblenz - Wikitravel

Koblenz is a city in the Middle Rhine Valley in the state of Rhineland-Palatinate in the Federal Republic of Germany. **Koblenz** has about 110,000 inhabitants; the ...

wikitravel.org/en/Koblenz

Koblenz 2016: Best of **Koblenz**, Germany Tourism - TripAdvisor

Koblenz Tourism: TripAdvisor has 15,570 reviews of Koblenz Hotels, Attractions, and Restaurants making it your best **Koblenz** resource.

tripadvisor.com/Tourism-g187391-Koblenz_Rhineland_Palatin...

Koblenz

koblenz.de

Koblenz, also spelled Coblenz or Coblence, is a German city situated on both banks of the Rhine at its confluence with the Moselle, where the Deutsches Eck and its monument are situated. More at Wikipedia

Related Topics

Koblenz Category

University towns in Germany

Populated places on the Rhine

55 Hotels in **Koblenz** AD

Best Price Guarantee! Book your Hotel in **Koblenz** online.

booking.com/Koblenz-Hotels

Send feedback

Figure 1: Gaze-controlled browser interface

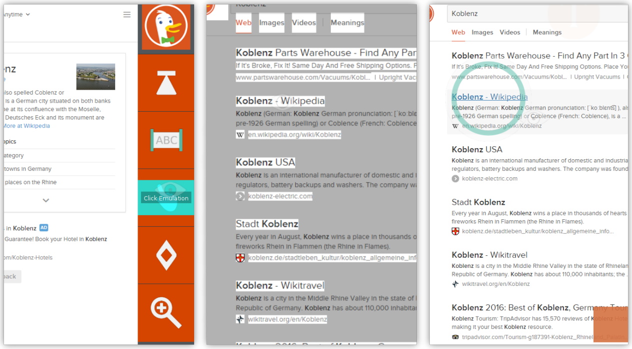


Figure 2: Navigation in Web. Left: Click emulation activated. Middle: Links are highlighted and page is zoomed continuously. Right: Click is performed.

To overcome the challenges, we present an open source framework to adapt Web interfaces for gaze interaction, where the input events (which are typically composed of mouse and keyboard interactions in generic applications) are revised to eye movements. We approach the problem of gaze-based browsing as a unifying framework of Web engineering and interface design. We identify the interactive elements of webpages via unsupervised extraction of DOM nodes, and represents these elements with explicit/implicit indicators to be accessed by eye gaze inputs.

The Web browser environment is built upon the *Chromium Embedded Framework*[1] (CEF), which provides an appropriate architecture to enhance the Web with interfaces for gaze interaction [6]. Adapting the interface is primary for the success of eye-based interactions [7, 9]. In this regard, the proposed Chromium based framework provides more utility and control to include eye gaze events in the browser, rather than building browser extensions. It has several advantages, e.g., we can receive webpage as pixels that give a complete power over webpage rendering, which is essential for dynamic interface adaptations for gaze interactions. Complex applications such as a gaze-based browser are also easier to maintain in C++ implementation with CEF. Moreover, it is convenient to integrate new devices in the framework (for browser extensions it is quite unnatural to connect to devices via JavaScript, and to call native system dependent libraries). The proposed framework is available as an open source system[2], which also provides in-depth information of the architecture.

2. EYE-CONTROLLED BROWSING

To provide a smooth and reliable eye-controlled browsing experience, we integrate the visual appearance and control functionality of webpages in an eye tracking environment. More specifically, our approach combines, webpage element extraction and eye gaze emulation of conventional input devices within the browser. For webpage interaction, the system examines the location of selectable objects on webpages, such as text input, hyperlinks, scrollable overflows, edit box, etc. and applies gaze-controlled interface paradigms for straightforward handling. Figure 1 shows an eye-controlled browser application developed using proposed Chromium based framework. The interface was developed through user centered design for eye interaction [9], i.e., interface components such as size, shape, appearance, feedback etc. that are vital to enhance eye tracking accuracy for input control [6].

The interface design of the browser (Figure 1) can be divided into three components. On the left, the Web panel covers common actions like going back and forward or opening the tab overview of the browser. The Tab panel on the right side of the interface represents the possible actions on the current webpage like text selection, click emulation or to enable automatic scrolling. The central part of the interface is the Web view containing the actual webpage, rendered by the underlying Chromium project. Interactive overlays are added for input fields within the page. The T-letter inside a circle representing the text input field to submit query on *DuckDuckGo*[3] search page.

A scrolling sensor is displayed on the bottom of the page indicating the control to scroll the page down (additional sensor on the top would appear to scroll up). The longer a user focuses on the sensor, the higher is its penetration and scrolling is accelerated towards a maximum speed. A vertical progress bar (visually embedded on the scrolling sensor) provides instant feedback of the current scrolling status. Furthermore, the user can switch on the automatic scroll functionality (and the dedicated scroll sensors would disappear) for more smooth and natural reading experience. The automatic approach is also the default behavior for scrolling within the page (e.g., Facebook chat box). We detect DOM elements that overflow, and imply automatic scrolling, since the space is too limited for additional manual scrolling sensor fields.

A significant aspect of a Web browser is hyperlink navigation. However, due to the potentially dense link structure of webpages, gaze-based link clicking suffers from the eye tracker's accuracy limitations. In the proposed browser, this accuracy problem is tackled using continuous zooming behavior. A click is initiated with the activation of the *Click Emulation* button in the Tab panel, see left image in Figure 2. Zooming process starts with any gaze coordinate upon the Web view, as displayed in the middle image of Figure 2. The zooming is centered at the gaze position on the webpage. When the zoom level is over a certain threshold, a click is performed at the focused page coordinate. A visual

[1] https://bitbucket.org/chromiumembedded/cef

[2] https://github.com/MAMEM/GazeTheWeb

[3] https://duckduckgo.com

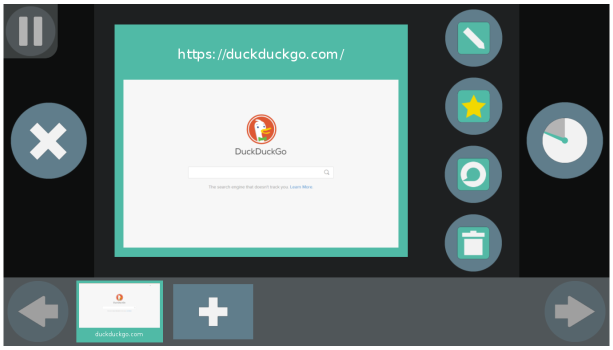


Figure 3: Management of tabs

feedback is provided by a shrinking circle around the click, as visible in the right most image of Figure 2.

The browser not only supports the efficient interaction with the webpage, it effectively supports all essential browsing menu operations like history, bookmarks and tab management for a wholesome surfing experience. Figure 3 shows the tab overview interface with the possibility of editing the URL, bookmarking, reloading, removing the tab, or opening new tabs. The detailed demonstration of the browser functionality can be viewed online[4].

3. THE UNDERLYING FRAMEWORK

Modern Web browsers are designed as high performance frameworks to cope with the task of interaction and rendering of numerous complex webpages. The current major version of CEF makes use of a multi-process and multi-thread approach to deliver a stable and fast user experience. Every tab runs in an independent process (*Render Process*), including instances of the Blink engine for rendering and the V8 engine for JavaScript handling. They are connected to a single *Main Process*, which handles user input, controls the tabs and performs rendering of the general user interface. If one tab is loading or its JavaScript execution crashes, the other tabs and the user interface keep running and are able to react to further user input. Figure 4 showcases the structure of our CEF based framework. The Mediator interface connects the *Visual Browser* part with our *CEF Implementation*. This interface passes commands like opening a new tab, changing a URL or input emulation to the corresponding CEF structures within CEF Implementation and provides DOM nodes of interest and pixels of rendered webpages. The Visual Browser creates the window as output for the OpenGL context, connects to eye tracking devices via dynamic linking into the vendors' SDK at runtime, handles the gaze-controlled user interface and eventually draws the pixels of the webpage on the screen.

3.1 Data Flow

Various data instances are transferred to the Visual Browser classes by the CEF Implementation to adapt Web browsing for eye-based input.

Renderer: We run CEF in offscreen-rendering mode, since no drawing onto a window directly by CEF is desired. The Renderer implementation hands the received pixel data of a rendered webpage to a WebView instance in Visual Browser, which fills them into an OpenGL texture object.

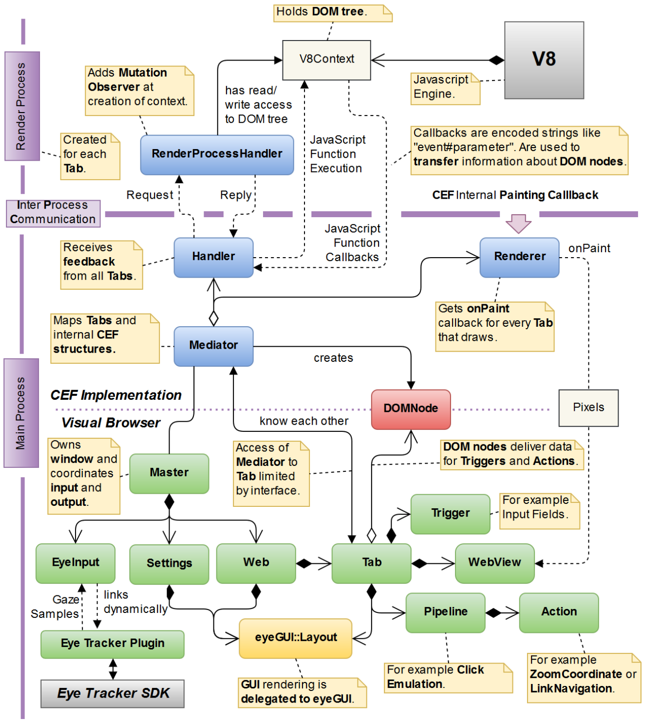

Render Process
Holds DOM tree.
V8Context
V8
Adds Mutation Observer at creation of context.
has read/ write access to DOM tree
Javascript Engine.
Created for each Tab.
RenderProcessHandler
JavaScript Function Execution
Callbacks are encoded strings like "event#parameter". Are used to transfer information about DOM nodes.
Request
Reply
Inter Process Communication
CEF Internal Painting Calllback
JavaScript Function Callbacks
Receives feedback from all Tabs.
Handler
Renderer
onPaint
Gets onPaint callback for every Tab that draws.
Maps Tabs and internal CEF structures.
Mediator
creates
Main Process
CEF Implementation
DOMNode
Pixels
Visual Browser
know each other
Owns window and coordinates input and output.
Master
Access of Mediator to Tab limited by interface.
DOM nodes deliver data for Triggers and Actions.
Trigger
For example Input Fields.
EyeInput
Settings
Web
Tab
WebView
Gaze Samples
links dynamically
Pipeline
Action
Eye Tracker Plugin
eyeGUI::Layout
Eye Tracker SDK
GUI rendering is delegated to eyeGUI.
For example Click Emulation.
For example ZoomCoordinate or LinkNavigation.

Figure 4: Architecture of proposed Framework

DOM Nodes: DOM nodes are useful in different scenarios of adapted user interaction. Extraction while the page is getting loaded and updated is desirable to provide a real-time interface that corresponds to the displayed webpage. Today's Web makes heavy use of dynamically loaded content, and hence simple polling of DOM tree parsing at a specific time of execution (like end of initial page loading) is not sufficient. We propose to use direct callbacks from JavaScript into C++ in combination with JavaScript-side observation of the DOM tree for dynamic changes. The Message Router interface of CEF provides direct callbacks of JavaScript into C++ by calling a predefined function in JavaScript with an argument that is then passed to a predefined method in the Main Process of the application. We encode the data into a string, where the single arguments are separated by a delimiter symbol.

This mechanism is combined with the *Mutation Observer*[5], which observes changes in DOM tree. It is appended to the root of DOM tree at V8 context creation on Render Process. When a change takes place in the DOM tree, a predefined JavaScript function is called while providing a list of *MutationRecords* that holds information about the changes. Type of change in DOM tree is then classified, DOM nodes of interest are added to a global list and C++ is called back with encoded minimal information about the change. As the string is received and parsed in Main Process, potentially more information about change are requested via *Inter-Process Communication* (IPC). The Render Process

[4] https://youtu.be/x1ESgaoQR9Y

[5] https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver

looks up to required DOM node information in the global list, since it has a direct access to V8 context of webpage, and then replies over IPC with requested information about the added DOM node. Afterwards, the DOM node structures in the framework are updated and shared with the corresponding Tab object in the Visual Browser part.

3.2 Data Usage

The extracted webpage data is employed in various situations to adapt browsing to eye gaze input.

- **WebView** receives pixels of rendered webpage from Renderer, as described earlier. The webpage is rendered onto a rectangular geometry with a custom OpenGL shader program. This is for example useful at click emulation to magnify the webpage with a continuous zooming effect to compensate the imperfect accuracy of eye trackers, as it can be simply implemented by a texture coordinate transformation.
- **Links** are used to highlight their bounding boxes while click emulation executes continuous zooming, and to correct click coordinates which are not on the extracted link but in the neighborhood of bounding box.
- **Text Input Fields** are detected to place gaze-controlled overlay buttons that call the virtual keyboard for text input. The text is filled in by JavaScript injection over the Mediator interface.
- **Fixed elements** are usually applied to static page elements like navigation bars. When a user fixates on static DOM elements, global automatic scrolling is avoided.
- **Overflow elements** are necessary to be scrolled to reveal the entire containing information. Therefore, we employ an automatic scrolling mechanism, which centers the coordinate of gaze within overflow elements.

There is a huge potential for further usage scenarios of extracted DOM nodes, i.e., the general availability of DOM nodes within the same programming unit is not only relevant for eye gaze input, but it provides a possibility of in-depth integration of next generation input devices (e.g., brain computer interfaces) in Web interaction.

3.3 Pipeline System

We have implemented different actions for gaze driven events that are applicable in various interaction scenarios, e.g., the continuous zooming is suitable for both click emulation and text selection. Hence, we propose a pipeline system that consists of independent modules, called actions. Here are some examples of currently available actions.

- **Keyboard** is an inclusive text display with simple editing features. It optionally takes an initial text as input, and outputs the collected text composited by the user of input and typed text. It successfully finishes if the user either hits the `ok` or the `submit` button.
- **ZoomCoordinate** holds the behavior described for continuous zooming in click emulation. Output is a coordinate on the webpage. Actions have limited control over Web view rendering, so this action does highlight the links while activation. This supports the selection process of the user by providing a visual feedback about clickable content.
- **LinkNavigation** takes a coordinate on the webpage and checks, whether the coordinate is upon a link. If the result is `false`, the distance to the extracted DOM links is measured. When one's distance is lower than a threshold, the coordinate is moved to the center of this link. Eventually, a left mouse button click is emulated at the coordinate.

A pipeline is a linear combination of one or more of the above-mentioned actions. When a pipeline is added, the first action in the pipeline is activated and executed. Once the action is finished and deactivated, the next action gets activated and so forth. Inherently the pipeline can be considered as a data stream that collects input data from the user while executing the desired action. For example the click emulation consists of first the ZoomCoordinate and second the LinkNavigation action.

3.4 Limitations and Challenges

We propose an efficient architecture to include eye gaze events in modern webpages. However, Web is complex and it brings several challenges due to the non-standardization or unintended use of Web technologies. Here we describe some of these limitations and future challenges to improve the interaction further.

Semantic Information

For a convenient experience with gaze input, we aim to provide different options to users, e.g., the text input action can apply the entered text to the input field or can be directly submitted for processing (e.g., search query). However in several instances input submission depends on the input in other fields, like a password cannot be submitted without username or the scenario of completing a registration form with multiple required entries. In such scenarios, deactivating the submit button or automatic transformation to a "next-text-input-field" function would provide a more user-friendly interaction. To accomplish this, we need precise semantic information about the input fields, which is hard to extract from the DOM tree. We could employ trivial rules, like in the scenario of username and password combination, search for DOM nodes with tag "password" and other input fields within the same form. However, a more robust approach is required to gauge the semantic information.

Text Input Field Density

Specific forms for registration or communication tools feature a high number of text input fields, which are represented by overlay icons in our framework. These overlay buttons have to be rendered at a minimum size in order to cope with the accuracy of the eye tracking device. Hence, in the scenario of complex forms, the buttons might overlap with each other and interaction becomes difficult and error prone. Hence we currently work on the design methods to resolve such issues with high-density of input fields.

Extensive use of CSS and JavaScript

Extensive use of modern Web technologies limits the success of DOM node extraction. A prominent example is the search input field on the Google page[6]. At the time of our assessment, there are at least two text input fields stacked onto each other for search box. It appears that one field is responsible to display suggestions in gray color and the other one is used for actual user input. It utilizes the CSS mechanism of z-value to advise the browser to render the actual input field in front of the other, which is not detectable in naive DOM node extraction. One has to do additional CSS property look-ups in order to find the text input field to be filled by keyboard. It is not just sufficient to check the z-value, since there may be multiple *real* text input fields and some covered ones, which are not accessible to a user but share the same z-value. At present we do in-depth investigation to resolve such issues for popular websites, however a universal solution is required to deal with these scenarios.

Additionally a major challenge arises from single CSS values, as they are not directly observable by the available Mutation Observer. Currently, we are able to track changes of the assigned CSS classes. An elaborate solution would be to fork the Blink engine code that handles the DOM tree, and cooperate native callbacks into the Render Process when any CSS value of a DOM node of interest is changed.

4. EVALUATION AND CONCLUSIONS

We conducted an experimental evaluation to quantify the performance and usability of the proposed system. We used OptiKey [1] as a benchmark, which is a state of the art open source tool for gaze interaction using the conventional approach of mouse and keyboard emulation by eyes. Eleven participants (4 female, 7 male) took part in the study, and they were asked to perform common browser tasks such as search, navigate, bookmark [5]. Tasks were identical for both systems, and the dwell time was configured to one second to negate any bias between the system. Eye movements were tracked with SMI REDn remote eye tracker with a sampling frequency of 60Hz, which was attached to the front of a 24 inch monitor. The experimental results indicate that the proposed system performs consistently better than OptiKey in not only task accomplishment, but also in terms of usability and cognitive load measures. The average time required to complete the specified task by proposed system was 252.18 seconds, significantly better than 424.15 second required for OptiKey (p-value 0.0023). Furthermore, in the SUS usability analysis our system reached the average score of 83.86, which is considered to be highest grade in SUS guidelines[7], while the OptiKey's score of 57.96 is well below the standard average. The participants also felt significantly less workload (measured by NASA-TLX test[8]) in the proposed system with an average score of 43.0, compared to OptiKey's score of 54.5.

The presented Chromium based framework offers the significant prospect of including gaze interaction in Web applications. Hence the user could perform all essential browsing operations by gaze in an easy and effective manner, as shown by the usability evaluation of our eye-controlled browser. We are currently working on further enhancements to include sophisticated gaze interaction features such as secure login/password entry, and better Web video interaction by customizing native HTML5 controls. The proposed framework comprises an extendable architecture, and in future we want to enhance its usability with additional modalities, i.e., integrating emotions into the browsing experience [8].

5. ACKNOWLEDGMENTS

This work is part of project MAMEM[9] that has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement number: 644780.

6. REFERENCES

[1] Optikey: Type, click, speak. https://github.com/OptiKey/OptiKey.

[2] R. Bates and H. O. Istance. Why are eye mice unpopular? a detailed comparison of head and eye controlled assistive technology pointing devices. *Universal Access in the Information Society*, 2(3):280–290, 2003.

[3] R. Biedert, G. Buscher, S. Schwarz, M. Möller, A. Dengel, and T. Lottermann. The text 2.0 framework: writing web-based gaze-controlled realtime applications quickly and easily. In *Proceedings of the 2010 workshop on Eye gaze in intelligent human machine interaction*, pages 114–117. ACM, 2010.

[4] R. Jacob and S. Stellmach. What you look at is what you get: Gaze-based user interfaces. *interactions*, 23(5):62–65, Aug. 2016.

[5] M. Kellar, C. Watters, and M. Shepherd. The impact of task on the usage of web browser navigation mechanisms. In *Proceedings of Graphics Interface 2006*, GI '06, pages 235–242, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[6] C. Kumar, R. Menges, and S. Staab. Eye-controlled interfaces for multimedia interaction. *IEEE MultiMedia*, 23(4):6–13, Oct 2016.

[7] M. Kumar. *Gaze-enhanced user interface design*. PhD thesis, Citeseer, 2007.

[8] A. Lugmayr and S. Bender. Free ux testing tool: the ludovico ux machine for physiological sensor data recording, analysis, and visualization for user experience design experiments. In *Proceedings of the SEACHI 2016 on Smart Cities for Better Living with HCI and UX*, pages 36–41. ACM, 2016.

[9] R. Menges, C. Kumar, K. Sengupta, and S. Staab. eyegui: A novel framework for eye-controlled user interfaces. In *Proceedings of the 9th Nordic Conference on Human-Computer Interaction*, NordiCHI '16, pages 121:1–121:6, New York, NY, USA, 2016. ACM.

[10] A. M. Penkar, C. Lutteroth, and G. Weber. *Eyes Only: Navigating Hypertext with Gaze*, pages 153–169. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[11] M. Porta and A. Ravelli. Weyeb, an eye-controlled web browser for hands-free navigation. In *Proceedings of the 2Nd Conference on Human System Interactions*, HSI'09, pages 207–212, Piscataway, NJ, USA, 2009. IEEE Press.

[12] B. Wassermann, A. Hardt, and G. Zimmermann. Generic gaze interaction events for web browsers. In *Proceedings of the 21st international conference on World Wide Web*. Citeseer, 2012.

[6] https://www.google.com

[7] http://www.measuringu.com/sus.php

[8] https://www.nasatlx.com

[9] http://www.mamem.eu