Voiceye: A Multimodal Inclusive Development Environment

Bharat Paudyal
DMT Lab, BCU
bharat.paudyal@bcu.ac.uk

Chris Creed
DMT Lab, BCU
chris.creed@bcu.ac.uk

Maite Frutos-Pascual
DMT Lab, BCU
maite-frutos@bcu.ac.uk

Ian Williams
DMT Lab, BCU
ian.williams@bcu.ac.uk

ABSTRACT

People with physical impairments who are unable to use traditional input devices (i.e. mouse and keyboard) are often excluded from technical professions (e.g. web development). Alternative input methods such as eye gaze tracking and speech recognition have become more readily available in recent years with both being explored independently to support people with physical impairments in coding activities. This paper describes a novel multimodal application ("Voiceye") that combines voice input, gaze interaction, and mechanical switches as an alternative approach for writing code. The system was evaluated with non-disabled participants who have coding experience (*N*=29) to assess the feasibility of the application in writing HTML and CSS code. Results found that Voiceye was perceived positively and enabled successful completion of coding tasks. A follow-up study with disabled participants (*N*=5) demonstrated that this method of multimodal interaction can support people with physical impairments in writing and editing code.

Author Keywords

Eye gaze tracking; Speech recognition; Assistive technology; Programming tools.

CCS Concepts

• Human-centered computing~Interaction design~Systems and tools for interaction design

INTRODUCTION

Programming involves multiple activities including designing, writing, debugging, compiling and editing code [39, 58]. In order to complete programming tasks, developers typically have to use a keyboard and mouse as the dominant input paradigm for controlling systems [28, 33, 52]. People with physical impairments (who are unable to use these input devices) can therefore be excluded from development work and the opportunity to have technical careers in this area. Eye gaze tracking and speech recognition are two technologies available to physically impaired users that can support interactions with different systems [7, 21, 36]. Both technologies have evolved rapidly in recent years and have been explored independently for supporting development work [1, 10, 13, 49].

For instance, the use of speech input has received recent interest as an alternative input method within programming environments [41, 52]. Tools such as these are presenting new opportunities for programmers with physical impairments through reducing the dependency on a traditional keyboard [9, 46, 51]. However, voice based coding tools are typically tailored for experienced coders and are rarely evaluated with physically impaired users [52]. Speech input also presents some limitations in a coding scenario through known challenges such as accurately detecting speech input [9, 28, 52, 64]. Moreover, little work has investigated the optimal techniques for selection and pointing tasks using a purely speech based approach [53].

Similarly, studies have investigated the use of gaze as a pointing device in coding environments [27, 50, 56]. There has also been substantial activity around controlling traditional and novel keyboard layouts via different gaze techniques [8, 44]. In contrast to speech recognition, these methods can potentially provide users with more control when entering text or selecting different application features (thus reducing the impact of incorrect speech input). However, whilst gaze interaction can make systems somewhat accessible, it has known limitations around the selection of small targets [19, 65], the well-known Midas touch issue [34, 36], and slow typing speeds [37, 40].

Several researchers have stated that the disadvantages of each modality (speech and gaze) can be reduced when both are combined into a multimodal approach [14, 42, 53, 65]. Initial work has explored the combination of both methods in different applications such as word processors [8] and creative software [36], as well as for controlling desktop

DIS '20, July 6–10, 2020, Eindhoven, Netherlands

ACM ISBN 978-1-4503-6974-9/20/07…$15.00
https://doi.org/10.1145/3357236.3395553

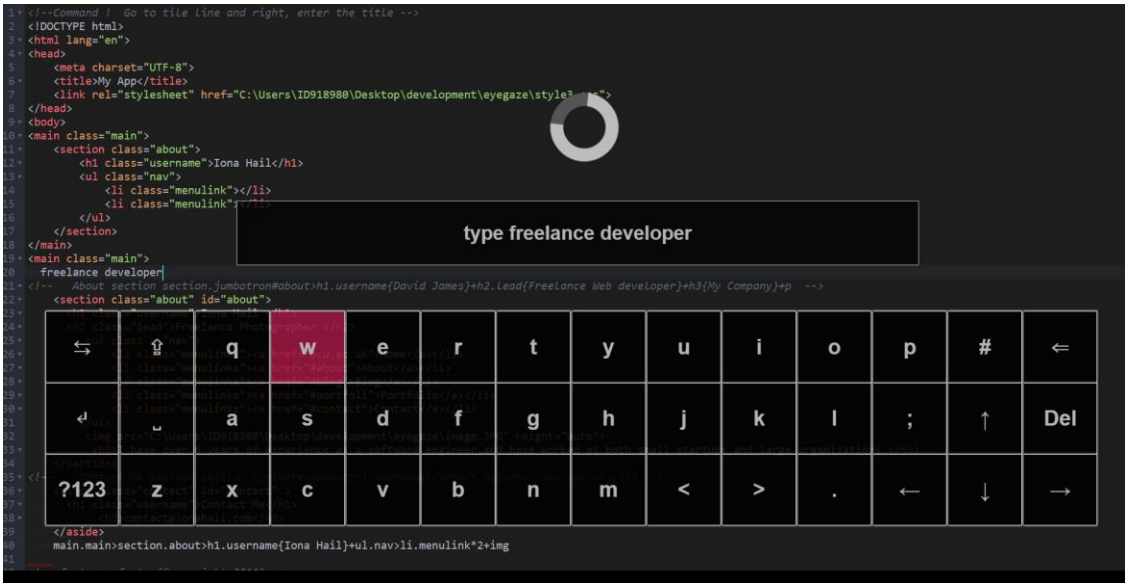

Figure 1: Screenshot of the Voiceye interface when the speech recognizer has been activated. The spinner indicates that the recognizer is listening for speech input. The text "type freelance developer" has been issued by the user in this example. The user's gaze is also hovering over the "w" key (resulting in visual feedback through the red background).

environments [35, 53] and web browsing [55]. This type of combined approach presents numerous opportunities to make interactions more accessible for people with physical impairments, although there has been a lack of work exploring this form of multimodal interaction. It therefore remains unclear what the optimal complimentary roles are for each input method when utilized to support coding activities. Moreover, the use of additional controls (e.g. mechanical switches commonly used by people with physical impairments) could also further support a combined gaze and speech approach, although we have little understanding around the feasibility of a system integrating multiple methods of input in this context.

To address the lack of work in this area, we present a new development application ("Voiceye") that combines eye gaze, voice, and mechanical switches as an approach for writing HTML and CSS code. The system uses voice input for verbal commands such as selecting, navigating, and removing code – as well as for dictating longer forms of non-code text (e.g. comments). Gaze is used to provide a more controlled approach to write code via an on-screen keyboard – to address issues with slow typing speeds we integrated Emmet [26] as a novel approach that enables users to write HTML/CSS code via a shorthand notation. Emmet is also widely used in industry to support more efficient coding workflows. The application was evaluated in an initial user study with non-disabled participants (N=29) where results have shown that it was perceived positively and enabled successful completion of different coding tasks. A follow-up study was conducted with five physically impaired programmers where the application again received positive feedback and enabled participants to write code during a web development activity.

The primary contributions of this work are threefold: (1) a novel multimodal coding environment enabling people with physical impairments to write code, (2) a user study demonstrating the usability of the system and new insights into this form of multimodal interaction, and (3) validation of this approach highlighting that people with physical impairments can effectively write code.

RELATED WORK

Coding via Speech Interaction

Voice based coding methods have been explored to investigate their potential to support writing code [11, 23, 49, 52]. Researchers have examined taking natural language or pseudo code as voice input and converting it into code in applications such as NaturalJava [48], VoiceCode [23], and VoiceGrip [22]. More recently, Gordon [29] created a new programming language where users provide vocal input in the form of pseudo code which is then converted into full syntax. Ayub and Saleem [3] developed a system that enabled users to generate C++ code through verbalization of the language's syntax and semantics (i.e. "see out" generates the code "*cout*<<"). Patel and Patel [46] also used the same approach to explore the potential for Java programming via speech control.

In terms of HTML and CSS coding, Modak et al. [43] and Chadha et al. [16] investigated the generation of webpages through natural language speech input. However, their system has some limitations in terms of including attributes such as "class" and "id", as well as other common HTML5 elements (e.g. figcaption, header, etc.) Similar work has been explored by Bajwa et al. [4], although their system only supports the creation of HTML forms via text.

Other approaches have investigated the potential of dictating code directly [9, 1, 49, 60, 52]. For instance, Begel and Graham [10] developed a code dictation tool (SPEED) to support people with Repetitive Strain Injury (RSI) in writing code [9]. Similarly, Rosenblatt et al. [52] investigated dictation of code with people who have upper limb mobility impairments. Moreover - VoiceCode [1], TalonVoice [60], and a new approach developed by Rudd [49] are existing vocal programming systems targeted for experienced programmers with RSI. These applications and approaches hold potential for disabled coders, although they might be cumbersome for novice programmers as they have to learn new non-natural voice commands (e.g. in VoiceCode, users have to utter the phrase '*snake shark attribute accessor*' to type *attr_accessor*).

Coding via Gaze Interaction

There has been significant research activity around gaze interaction and its potential in creating accessible applications for people with physical impairments [7, 36, 38]. In particular, there has been substantial work around different eye typing techniques [17, 40], as well its use in supporting the selection of small targets [6, 42, 57], document navigation [47], drawing [31, 32], and web browsing [38, 47]. However, little work to date has focused on using gaze to support programming activities.

Initial work has explored its use as a pointing device in coding environments - for instance, EyeDe [27] is a gaze supported IDE that provides support for features such as navigating to a function, tabbing between documents, and opening source files. Similarly, Radevski et al. [50] developed EyeNav which combines gaze and keyboard shortcuts to support common development activities related to code selection, page scrolling, and single character movement. CodeGazer [56] is a gaze based system that supports users with coding navigation tasks such as "go to definition" and "find all usages"

Whilst there has been significant research exploring how to make standard gaze typing tasks more efficient [37, 40, 44], there has still been little work investigating the optimal approaches for writing code. Moreover, mainstream development environments (e.g. Visual Studio Code [62]) contain lots of small icons and interface elements that are difficult to select via gaze interaction [19, 42]. The majority of research studies to date investigating the use of gaze input for navigating, writing, and selecting code have also not involved the use of physically impaired coders. Further work is therefore required to explore accessible development approaches for disabled coders.

Coding via Gaze and Speech Interaction

A number of researchers have suggested that the combination of gaze and voice can be an intuitive interaction solution in a number of scenarios [25, 30, 36, 55]. Research studies have started to explore the combination of voice and gaze in non-programming fields [8, 14, 36, 53, 55] where gaze is typically used as a pointer for interaction and speech input for issuing commands and performing interface selections. In terms of multi-modal interaction in a programming environment, TalonVoice [60] uses both gaze and voice to enable people with limited or no use of their hands to write code, although it is mostly targeted for experienced programmers. In this application voice is used for typing code (as well as issuing commands) whereas gaze is used for cursor control and zooming into areas of code. Apart from TalonVoice, which also lacks academic evaluation, no other studies have been conducted exploring the combination of voice and gaze to support development work. Moreover, no studies have investigated the potential of additional input methods to further support a combined speech and gaze approach (e.g. the use of mechanical switches for performing selections).

APPLICATION DESIGN

To address the lack of research exploring the potential of multimodal interaction approaches to facilitate coding activities for disabled users, we developed a code editor (Voiceye) that can be operated using the combination of gaze, speech input, and mechanical switches. Voiceye is a desktop-based application developed using Electron.js [24] that enables writing and editing HTML/CSS code via gaze interaction, whilst voice input is used for dictating long text (e.g. comments) and performing commands such as selecting, deleting, and navigating code. It is built on top of CodeMirror [18] – an open source JavaScript based code editor which provides standard features such as syntax highlighting, search, replace, and code indentation.

Voiceye consists of four components: an automatic speech recognizer (using Microsoft's Azure Speech service [59]), a rule-based syntax grammar, an onscreen keyboard (controlled via gaze input), and two mechanical switches – one for performing selections on buttons within the interface (e.g. a virtual keyboard key) and the other for toggling activation of the voice recognition system. As the user speaks, the prototype converts speech to text and performs a check against the rule based syntax grammar listed in Table 1. Appropriate actions are then triggered based on the spoken command. The onscreen keyboard allows users to type code via fixating on characters of interest and then completing the selection by pressing one of the designated mechanical switches.

The main interface was informed through the design of existing mainstream development environments (e.g. Visual Studio Code [62], Atom [2], Brackets [12], etc.). These applications typically display the line numbers by default on the left-side of the interface with the code included in the main interface area (with features such as syntax formatting enabled). We adopted a similar interface, along with a theme similar to the default one used in Visual Studio Code (Figure 1). The QWERTY keyboard layout is the most common one used in eye typing studies [37, 44]. This layout is therefore integrated into the application with keys that are 110x110px in size to support more comfortable

Tasks	***Speech Commands***	***Description***	***Example Code / Utterances***	***Example Output***
Code Entry	"Type"	Entering new text.	■ *Speech:* "Type welcome"	welcome■
Navigation	"Go to {line number}"	Navigating between the lines of code.	*Speech:* "Go to 17"	Cursor placed at start of line 17.
	"Left/Right"	Navigate one position left/right.	<main■class="main"> *Speech*: "left"	< ■main class="main">
	* "Left/Right {number}"	To navigate x positions left/right.	padding: 4px ■ 2px 3px 4px; *Speech*: "Left 2"	■padding: 4px 2px 3px 4px;
	"Up/Down"	Navigate one line up/down.	8. 9. ■ User 10. *Speech*: "Up"	8. ■ 9. User 10.
	"End of Line"	Navigate to the end of current line.	■ User *Speech*: "End of line"	User ■
Selection	"Select"	Select one /multiple elements of code.	padding: 4px■2px 3px 4px; *Speech*: "Select Select"	padding: 4px 2px 3px ■ 4px;
	"Select {line number}"	Select a single line.	8. ■<nav> 9. header 10. </nav> *Speech*: "Select 9"	8. <nav> 9. header ■ 10. </nav>
	"Select {line number} to {line number}"	Select multiple lines.	12. ■ 13. … 20. *Speech*: "Select 12 to 20"	12. 13. … 20. ■
	"Select {property}"	Select attributes and values.	<img■src="abc.png" height="20px"> *Speech*: "Select property"	<img src="abc.png"■ height="20px">
Deletion	"Delete line {line number}"	Delete a single line.	12. ■<nav> 13. header 14. </nav> *Speech*: "Delete line 13"	12. <nav> 13. ■</nav>
	"Delete line {line number} to {line number}"	Delete multiple lines.	11. <section> 12. ■ 13. … 20. 21. </section> *Speech*: "Delete line 12 to 20"	11. <section> 12. ■</section>
	"Delete selected"	Delete highlighted code.	padding:4px 2px 3px 4px; *Speech*: "Delete selected"	padding: 4px 2px 3px■;
Add Comment	"Comment"	Add single line comment.	■padding: 4px 2px 3px 4px; *Speech "comment"*	*/*padding: 4px 2px 3px 4px;*/* ■

Table 1: List of main vocal commands used in Voiceye. "Vocal Commands" column contains key words/terms that are actually spoken by users. Underlined text in "Example Code / Utterances" column includes example code to demonstrate how each speech command functions. The action performed through each speech command is visualized in the "Example Output" column. ■ denotes the position of cursor. * denotes new commands added after first user evaluation.

selection via gaze. The keyboard also included standard delete, backspace, space, enter, shift/capslock, and tab keys. A "char" key was also provided ("123?") to toggle between letters, numbers, and special characters. Arrow keys are included that allow users to navigate through single characters of text (speech provides word level navigation). Early usability testing found that placing the keyboard at the bottom of the screen resulted in the bottom keys being problematic to select via gaze – 100px of padding was therefore applied to the bottom of the keyboard to make all keys easier to select.

To address the issues of slow typing speeds via gaze, Emmet [26] was integrated into the system which allows users to write HTML and CSS code using a shorthand notation that is then expanded to full syntax. For example, the user can type "*div#menu.side*" to generate the code *"<div id="menu" class="side> *". A number of approaches were also considered for performing selections (i.e. dwell time, speech commands, gaze gestures, etc.), but we opted for mechanical switches that are commonly used by people with physical impairments [20]. There have been lack of studies investigating the use of switches as a selection approach in a multimodal gaze application, so this approach also presents an opportunity to explore users' perceptions of this type of interaction approach.

Speech input can be activated using the designated switch – upon selection a semi-transparent text area is displayed above the keyboard, along with an animated spinner. When the user speaks, the text the system has recognized is displayed to the user to provide instant feedback. The spinner provides a visual cue that the system is processing the speech input. These interface elements were added after initial usability testing on an earlier version where users expressed frustration at not knowing if the system had detected their input correctly. This typically resulted in a delay to see if the system performed the correct action or whether the user had to repeat the command.

The choice of vocal commands (Table 1) were informed through Rosenblatt et al.'s [52] Wizard of Oz study with coders, as well as their final choice of commands integrated into the VocalIDE system. Additional commands were also included such as "*open/close keyboard*", "*clear*" (to clear any highlighted words), and "*undo/redo*".

EVALUATION

An evaluation was conducted with non-disabled participants to investigate the feasibility of multimodal interaction in a coding environment. This was crucial to ensure that the multimodal approach was viable and appropriate before moving onto evaluating the system with disabled coders. A key requirement for using the application is that all users are able to use gaze to type, voice to control the interface, and operate switches for selection and triggering the voice recognizer. It was therefore felt that a first study with non-disabled participants would provide an important and relevant insight into the use of this multimodal approach for coding purposes. It also provided an opportunity for identifying areas where future improvements could be made through iterative development work (prior to conducting evaluations with physically impaired developers).

Participants

29 participants (two females) from a population of university students were recruited with ages ranging from 19 to 45 years (M=27.9; SD=7.87). Nine were native English speakers, whilst other native languages included Urdu, Bengali, Malay, Romanian and French. 13 participants wore corrective lenses (10 glasses). Participants completed a standard consent form before the test and were not compensated. They were asked to self-assess their web development skills, as well as experience in using alternative input methods for interaction with computers. Eight participants had prior experience in using an eye tracker, whilst 20 had previous experience of using speech recognition. 25 participants had some coding experience with an average of 1.7 years (SD=4.02) whilst the other 4 had a basic understanding of HTML and CSS.

Apparatus

The study was conducted on a Windows 10 laptop (Intel® Core(TM) i3-7100U CPU and 8GB RAM) using an external 23-inch LCD monitor with 1920x1080 resolution. Voiceye was installed on the machine - the Eye Tribe eye tracker [61] was placed in front of the monitor on a tripod (approximately 60cm from participants' eyes). The Eye Tribe provides an average accuracy of 0.5 to 1° of visual angle and an operating range between 45-75cm. The laptop's built-in microphone was used to detect speech input. Two 65mm Jelly Bean switches were placed in front of the monitor and eye tracker (Figure 2) – the one on the left was used to trigger keyboard selections and the one on the right for activating the speech recognizer.

Procedure

Pre-Test: Participants were provided with an information sheet containing details about the study and asked to provide informed consent. They were also asked to complete a pre-test questionnaire to collect demographic information, as well as details on their level of experience with software development and use of alternative input methods. Participants were given a demonstration of the prototype and encouraged to ask any clarifying questions. A nine-point calibration process was then performed using the Eye Tribe sensor. After successful calibration, participants were asked to practice with the application for 5-10 minutes to ensure that they could comfortably control the interface. During the practice session they were asked to write HTML and CSS code, navigate to different line numbers, select and delete code, and edit syntax errors.

Main Test: After the practice session, participants started working through the main tasks. The tasks chosen were designed and categorized based on the web development

scenarios utilized by Rosenblatt et al. [52]. The main task categories included Adding Code [ADD], Selecting Code [SELECT], Deleting Code [DELETE], and Editing Code [EDIT]. Each category consisted of 16 tasks of which 8 were related to HTML and the remaining 8 were focused around CSS (64 in total). The EDIT tasks were informed by the highest occurring HTML/CSS syntax errors made by computing majors [45].

The ADD tasks for HTML included adding a new element (e.g. <h1></h1>), a new element with single or multiple properties (e.g.), creating elements with child elements, writing HTML manually without using Emmet, adding some paragraph text (e.g. Welcome to my Portfolio), and creating a freeform comment. ADD tasks for CSS included adding new empty styles for classes (e.g. ".div {}"), styles with single and multiple properties (e.g. "color: #000; background: #FFF"), styles targeting children of an element (e.g. ".div>span"), writing CSS manually without Emmet, and standard freeform comments.

The SELECT tasks for both HTML and CSS involved selecting elements/styles, lines, properties of elements/styles, blocks of code (i.e. spanning multiple lines), and specific words within a block of text. DELETE tasks required removing elements/styles, lines, element/style properties, a block of code, specific words within a block of text, and a comment. Finally, EDIT tasks included fixing typographical mistakes, addressing unclosed element pairs, correcting comment syntax, and fixing misidentified or "confused" constructs (e.g. declaring <h1> instead of <title> within a header).

Participants were initially shown the "starting" code on paper that would be seen upon starting the task. The final completed code snippet was also shown to participants (on paper) in order to help them clearly understand what code or updates needed to be made. The researcher would outline the actions required to complete the task – voice commands and Emmet code related to the task categories were also provided to participants.

Once they made the necessary updates and verbally stated that they have completed the task, the researcher used a keyboard shortcut to move onto the next task. Once all the tasks for a particular category were completed (e.g. ADD tasks) participants then moved onto the next category and the process was repeated until all tasks were completed. The task categories and tasks within each category were randomized to reduce the potential impact of order effects. The start time, end time, vocal commands, and gaze actions (i.e. buttons clicked, position of gaze, etc.) were logged for later analysis. Our aim in designing the study was not to create a highly controlled evaluation, but instead to provide structured tasks that encouraged participants to gain experience in using the application.

Measures

Task Completion Time: Task completion times were measured in milliseconds from when participants started each task (i.e. after the researcher initiated the task via a keyboard shortcut) until the task had been completed. This measure was included to provide an *indication* of how long coding tasks took using an alternative multimodal approach.

Usability and Cognitive Workload: Perceptions of usability were measured through use of SUS [3] administered at the end of the study. NASA-TLX [54] was used to rate perceptions of workload.

Post-Test Questionnaire: To obtain qualitative feedback, participants were presented with an online survey with questions focused around their perceptions of using a multimodal approach for writing code, their experiences in using speech for issuing commands, how they found the use of gaze and Emmet for writing syntax, and any suggestions for future updates.

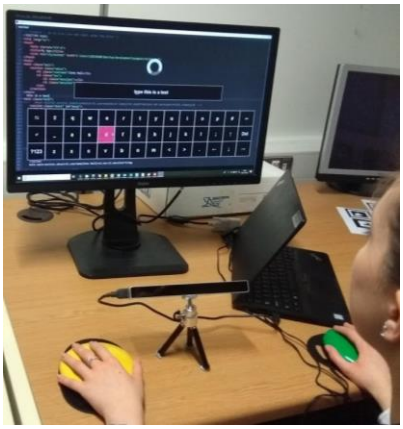

Figure 2: A participant performing the usability evaluation

Results

Task Completion Times

All participants completed the tasks with an average time of 41.10 min (SD = 12.04 min). The ADD tasks for both HTML (7:15 min – SD = 2.10 min) and CSS (8:11 min – SD = 4:41 min) took the longest to complete, along with EDIT HTML tasks (7:48 min – SD = 2:57 min). The average completion times for the other categories ranged from 2:30 min (SD = 0.46 min) for SELECT CSS to 4:58 min (SD = 2:06 min) for the DELETE HTML category.

Usability and Workload

The Voiceye application received an average SUS score of 68.1 (SD = 20.8). The score can be labelled as OK according to Bangor et al. [5]. An average NASA-TLX score of 42.0 (SD = 20.1) was received for the application indicating the prototype has a "somewhat high workload".

Survey Analysis

Speech Interaction: 20 participants provided positive responses around the use of voice for controlling the system

- comments focused around the voice being accurate, quick and easy to use: *"… the speech controlling system was efficient and easy to use*" (P15). Seven participants commented that some of the routine commands were not recognized and they had to sometimes repeat commands on multiple occasions. P28 highlighted that they preferred speech over gaze: "*it was much quicker, more reliable and allowed for shortcuts (such as end-of-line etc.) … with a few extra commands I would not need to rely on the gaze-based input as much*". However, P14 commented that they found the interaction difficult as *"…your eye would be focused on the editor, not the prompt coming back so it would take time to work out it had mis-heard you, then break your focus look at what happened and repeat*".

Gaze Typing: 16 participants provided positive comments about typing code via gaze stating that it was easy and intuitive to use: *"…it was reasonably quick and the keyboard layout was intuitive*" (P28). Five participants commented they initially found it challenging, but that they found it easier over time: *"…it was harder in the beginning, but I got used to it pretty quickly*" (P22). Seven participants commented that typing via gaze was "hard" or "difficult" including three participants who stated accuracy and calibration issues made the experience "frustrating" (P27, P29, P30). 20 participants stated positive comments on the use of Emmet when writing code via gaze. These positive statements tended to emphasize that the approach was fast and simple to use: *"…it was really easy as it did not take any time to write the code manually*" (P4). Six participants again commented that calibration and accuracy issues influenced their coding experience via gaze: "*it was a good way of speeding up the workflow, but the eye tracker's low accuracy made it tricky to be consistent.*" (P28).

Overall Multimodal Approach: 19 participants provided positive responses around their experience of using a multimodal approach in a coding environment. These participants described the approach as "simple" and "easy" to use: "*my overall experience was good, it was easy to use and the functions were simple…*" (P15). Further comments re-emphasized participants' overall positive perceptions of the prototype and ability to complete the tasks – for example, P17 commented that *"…there is potential for this tool to aid programmers in general; programmers using multiple screen at home may benefit particularly from talking to one screen and type-code on another screen - allowing programmers to sift through data files/code*".

Future Improvements: Four participants suggested including more voice commands to navigate around syntax: *"… the 'right' and 'left' options were very tedious to use. It might be useful to have some sort of feature which allows you to move several spaces in the left and right direction at a time*" (P6). Similarly, P28 commented that a larger range of commands would be useful *"… allowing for typing individual letters, allowing jumping to specific columns within a line and simplifying repetitive navigation tasks (such as moving to words in the middle of lines)*". Eight participants also emphasized the importance of ensuring the gaze system is accurate and comfortable to use.

FOLLOW-UP STUDY

A follow-up study was conducted with five users with physical impairments to explore their experiences in writing HTML and CSS using Voiceye.

Voiceye (Version 2)

To address feedback from the first study around enhancing syntax navigation, a new vocal command enabling users to jump multiple positions within a line was added (e.g. Left/Right x - "Left 5" would move the cursor five positions to the left from the current position). Additionally, several participants during the initial study commented on the system's misrecognition of some commonly used voice commands like "delete" and "select", so similar words were added to the speech recognizer and then mapped to the correct command (e.g. the system was often misrecognizing a "delete" command as "de'lite'" – this "incorrect" term was therefore added to the recognizer). Moreover, Wagner and Gray [64] previously highlighted that non-native English participants can yield more pronunciation errors, so the system was updated to support the selection of different English accents offered through Microsoft's Azure Speech service (to help further enhance recognition accuracy) [59]. Whilst most users in the first study were able to effectively control the system via the Eye Tribe device, we decided to integrate the Tobii 4C sensor [63] (which provides a higher level of accuracy and has a larger operating distance) to address calibration issues reported by some users.

Participants: Five participants were recruited through the support of the London RSI group [15] and existing links within the research team. Participants had a mean age of 23.6 years (SD=7.3 years) with four being diagnosed with Repetitive Strain Injury (RSI) and one with Cerebral Palsy. Participants were asked to self-assess their level of experience with web development skills and alternative input methods (Table 2). Participants had 2.4 years (SD=1.5 years) of web development experience and three participants had previously used speech recognition technology for development work, whilst none had prior experience of using eye gaze as an alternative interaction approach. Only one participant was a native-English speaker – the other native languages included Nepali, Lithuanian, Polish and Spanish. Two participants wore correctives lenses (all glasses).

Apparatus: The study was conducted on a Windows 10 laptop (Intel® Core(TM) i3-7100U CPU and 8GB RAM) with the screen resolution set to 1920x1080px. The Tobii 4C eye tracker was used and attached to the bottom of the screen via a magnetic connection. This sensor provides an average accuracy of 0.4 to 0.9° of visual angle and an operating range between 50-95cm. Two 65mm Jelly Bean switches were again used for completing gaze-based selections and for triggering the speech recognizer. The

ID	*Age*	*Impairments*	*Technical Experience*	*Challenges with Existing Tools*
P1	21 (M)	RSI (since 2017). Burning sensation in wrists and forearms.	*WD*: 2 years; *SRT*: 1 year; *EG*: None; *IDE*: Visual Studio; *AT*: Uses VoiceCode and vertical mouse.	Limited HTML/CSS support in VoiceCode; misrecognition of speech input.
P2	38 (M)	RSI (Since 2018); Pain in wrist, fingertips.	*WD*: 5 years; *SRT*: 1 year; *EG*: None; *IDE*: Web Storm; *AT*: Used Nuance Dragon once – uses mouse/keyboard.	Issues with speech recognition accuracy; experiencing severe pain when using keyboard/mouse.
P3	20 (M)	Epidermolysis bullosa (since birth); gets blisters from friction and extreme heat.	*WD*: 1 years; *SRT*: None; *EG*: None; *IDE*: Dreamweaver; *AT*: Previously used IntelliKeys – currently uses standard keyboard and vertical mouse.	Experiences pain when typing via keyboards (both standard and IntelliKeys).
P4	18 (M)	Mild Cerebral Palsy (since birth); affected mobility; stiff limbs; difficulty in typing.	*WD*: 1 years; *SRT*: None; *EG*: None; *IDE*: Dreamweaver; *AT*: None.	Difficulty typing on keyboard and using a mouse to control software (due to mobility impairments).
P5	21 (M)	RSI (Since 2019); Pain in left shoulder, numbness in left arm and fingers.	*WD*: 3 years; *SRT*: 1 year; *EG*: None; *IDE*: VSCode, Atom; *AT*: Uses TalonVoice.	No standardized voiced commands with TalonVoice; voice recognition accuracy issues.

Table 2: Participant Details - WD = Web development; SRT = Speech Recognition Tools; EG = Eye Gaze; IDE = Integrated Development Environment; AT = Assistive Technology

equipment was set up on a table and tailored to participants' needs (e.g. in terms of where the switches were placed).

Procedure: Institutional Review Board (IRB) approval was initially obtained for the study. Evaluations were conducted in either participants' home or work environment (P1, P2, P4 and P5) and at our research lab (P3). Participants were initially given an information sheet and asked to provide informed consent. A survey was then administered to collect demographic information, details of impairments, and information about their web development skills. Upon completion of the survey, participants were given a guided demonstration of the prototype by the researcher (via a keyboard). The eye tracker calibration process was then completed and participants were asked to practice using the system for around 5-10 minutes.

The main task was designed around a real-world development scenario in which a website had to be coded using HTML and CSS. Participants were provided with a screenshot of the website on paper to provide some context around the task (Figure 3). They were also given a sheet that included common voice commands, along with the Emmet commands required to complete the task. The Voiceye application was then started with a blank HTML and CSS document opened by default. Participants were asked to start working towards coding up the design using the multimodal interaction approach. Moreover, upon completing the activity, they were also asked to complete the main task separately using their existing assistive technology and code editor of choice (for comparative purposes). Three participants (P1, P2, P3) were able to complete this additional element of the study, whilst the other two participants preferred to leave this due to the potential discomfort it might cause them.

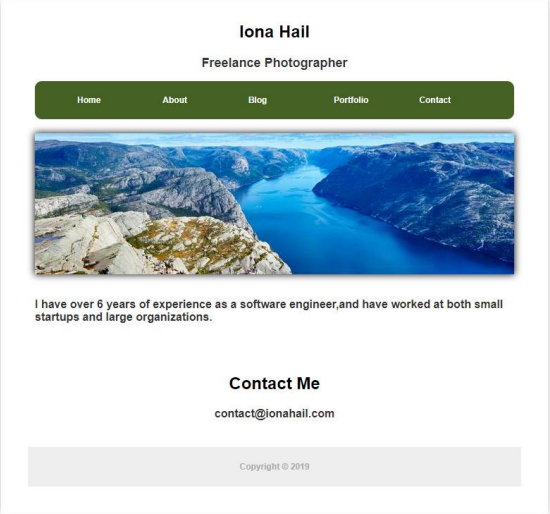

Figure 3: The design used for the follow-up study

Our aim with this study design was to give participants a realistic coding activity that encouraged them to explore and openly evaluate the application and interaction approach (as opposed to conducting a highly controlled study). There was no time limit set for the activity, although participants were informed that they could take a break or withdraw at any time if they experienced any fatigue, tiredness, or discomfort. Sessions lasted between 65-130 minutes and participants were then asked to complete a SUS form to assess the overall usability of Voiceye. They then completed an online survey with questions focused around exploring their experiences of using the system. Finally, a semi-structured interview was conducted to further investigate their perceptions of the application and the multimodal coding approach used.

Results

Usability (SUS): Voiceye received an average SUS score of 74.0 (SD = 4.6) which can be labelled as good and usable [5]. Four participants (P2, P3, P4 and P5) provided scores of 70 or over, whilst a lower score was provided by P1 (67.5), which can be labelled as "OK" in terms of usability.

Coding with Voiceye: All participants were able to utilize the features within Voiceye to write HTML and CSS code starting from a blank file. P1 and P5 were able to add more HTML elements (i.e. images, navigation, sections, etc.) whilst P2 spent more time creating styles for CSS classes to work on the presentation of the HTML (i.e. the code on the right-side of Figure 4). The code on the left-side of Figure 4 shows a screenshot of the HTML written by P1 including common page elements such as a header, navigation bar with a menu, images, and some longer text blocks. P1, P2, and P5 provided positive comments about the application in that it was simple, intuitive, and easy to use. Participants P3 and P4 were also able to utilize Voiceye to write code, although their final output typically incorporated fewer HTML and CSS elements. Time spent on the coding activity ranged from 15 minutes (P3) – 46 minutes (P2).

Coding with Existing Tools: P1 was able to write the majority of the HTML code, as well as creating multiple new CSS styles. However, P1 commented that he had to "stress" his voice when using VoiceCode often due to speech misrecognition – he was also only able to use one hand with the keyboard resulting in a slower typing rate. P2 was able to write the key components of a HTML document and create some new CSS styles, although he regularly required breaks due to experiencing pain when typing. It was also observed that both P1 and P2 had issues when attempting to edit code with each commenting they found editing incorrect syntax to be particularly cumbersome using their existing tools. P3 was able to produce markup for the basic components of a HTML document, although only wrote a single CSS style. This was influenced through P3's comment that he found it challenging when having to use multiple keys simultaneously (i.e. typing characters such as "<" where use of the shift key is required).

Voiceye vs. Existing Tools: P1, P2 and P5 reported that Voiceye was better than their existing coding tool and provided positive comments about the multimodal approach. In particular, P1 and P5 reported that the use of eye gaze and voice gave them the opportunity to select between the different input methods to fix errors when typing code, which was not present in their current voice based tool (i.e. TalonVoice and VoiceCode). Three participants (P1, P2 and P3) highlighted that Voiceye helped to remove the burden of using their hands when pressing keys when using a keyboard and mouse interaction approach *"...the prototype helps take load off hands…*" (P1). P4 stated that he has never previously been exposed to any assistive tools, but commented that Voiceye can be a viable approach to support him with coding.

Speech Control: All five participants utilized voice to perform coding activities and their comments focused around the approach being natural and easy to use, as well as commands being easily memorable. However, each participant still experienced some minor issues with the accuracy of the recognizer at times: "*…I know it will not recognise the programming languages, however the idea of using it as a command is really good approach. It worked well with slight issues (not recognizing...)*" (P5). P4 reported that his stutter got worse when trying to type longer text, but worked effectively with the main commands. Two participants (P2, P4) commented that they would like to see more voice commands (e.g. find/replace and select/delete for selecting and deleting specific word(s) within a line, rather than using left/right commands).

Typing Syntax via Gaze: Three participants (P3, P4, P5) commented that gaze was a fast and easy approach for writing code: *"… frankly, I thought it would be slow to type HTML … but with emit [sic] code, it was faster…*" (P5). Two participants (P1, P2) suggested that they would like to see improvements in the accuracy of the gaze interaction, although both were able to effectively use the system to write code. P2 and P4 highlighted that using the switches required some effort and could result in their hands becoming tired over prolonged periods of interaction.

DISCUSSION

Previous studies have independently explored the potential of gaze and speech input for writing code [9, 16, 27, 52, 56], although both approaches have been found to have strengths and limitations (e.g. challenges around the selection of small targets via gaze, speech recognition issues, etc.). Researchers have highlighted that the combination of speech and gaze could present a complimentary and more effective method of interaction [7, 36, 53, 55], although there has been a lack of research to date investigating the potential of this approach. This paper has presented a new system (Voiceye) that integrates these methods of interaction (in addition to mechanical switches) to enable physically impaired developers to write code.

The results from two user evaluations were positive and demonstrated the feasibility of combining gaze, speech, and mechanical switch interaction for supporting development work. The majority of participants from the first study found coding via multimodal input controls to be an intuitive and simple interaction approach. Physically impaired coders from the second study also made similar positive comments with several stating this approach provided advantages over their existing assistive methods of interaction. This combination of gaze and speech interaction (supported with switch controls) demonstrates how they can utilized together to help overcome some of the limitations of each technology. In particular, writing code via gaze interaction (using a common shorthand notation) helps to address some of the misrecognition issues around producing syntax via speech recognition. Similarly,

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <link rel="stylesheet" href="user_5.css">
    <title>my app</title>
</head>
<body>
<main class="main">
<section id="about" class="about">
<h1 class="usrname">Iona Hail

</h1>
<h2 class="lead">Freelancer photographer</h2>


<ul class="nav">
    <li class="menulinks"><a href="">home</a></li>
    <li class="menulinks"><a href="">abou</a></li>
    <li class="menulinks"><a href="">blog</a></li>
    <li class="menulinks"><a href="">portfolio</a></li>
    <li class="menulinks"><a href="">contact</a></li>
</ul>
<img src="image.jpg" alt="" />
  <h3>i have over 6 years of experience is a software developer,

</section>
<aside class="contact"></aside>
</main>
</body>
</html>
```

```css
body{
  margin: auto;
  font-size: 14;
  background-color: #fff;
  }

.main{
    width:700px;
    margin: 20px auto;
}
.section {
    padding:10px;
}
.section h1{
  color:#000;
    font-size: 20px;
   font-weight: bold;
    text-align: center;
}
.ns {
    width:500px;
    background:#ccc;
    border-radius: 10px;
    margin-bottom:20px;
}
.menu{
    display: inline-block;
    font-weight:bold;
    color:#fff;
    width:98px;
    padding: 10px;
    text-decoration: none;
}
```

Figure 4: The HTML script on the left is written by P1, while the CSS script on the right was coded by P2.

the use of speech input (using a fixed set of vocal commands) for enabling actions such as the navigation, selection, and deletion of code helps to overcome the issues of activating small targets via eye gaze. This new multimodal approach therefore provides key insights on the potential of combining speech, gaze, and mechanical switches for development work, as well as highlighting future challenges to be addressed.

From a wider perspective, this research highlights how multimodal interaction can support the control of systems, although there is a lack of work examining the combination of additional input methods to develop new assistive solutions (e.g. mid-air gesturing, head tracking, facial expression control, etc.). Also, as highlighted by P17 (during the first study), exploring the interplay of these new multimodal approaches with different external displays and configurations could hold much potential for both disabled and non-disabled coders (e.g. in terms of one display being optimized primarily for speech controlled activities, whilst another could support different tasks aligned to an alternative input method).

LIMITATIONS AND FURTHER WORK

A small number of participants across both studies experienced issues around the accuracy of the eye tracker when writing syntax (i.e. due to possible calibration drift) and in terms of speech recognition for activating commands (i.e. the recognizer misinterpreting commands). These are known challenges with gaze [42, 57] and speech interaction [10, 52, 64], although steps were taken to mitigate their impact through using large selection targets (110x110px) for gaze selection and through primarily using speech input for short focused commands. However, it is clear that for some users this method of interaction may present issues, so it will be important in future work to investigate making the interface more customizable to support the individual requirements of disabled coders (e.g. adjusting the size of targets, enabling users to define the words used for key commands, etc.). A further limitation was that the evaluation of the system focused primarily on writing and editing code. Whilst this is an essential first step, it will be important now to consider the multimodal interface design for other key programming activities such debugging, document control, source code navigation, and auto-completion. Several participants also commented that they would like additional vocal commands to be integrated into the system (e.g. for navigating and editing code through find/replace and select/delete commands).

CONCLUSION

We introduced a new application – Voiceye – that supports people with physical impairments in writing HTML and CSS code via a multimodal gaze, speech, and switch approach. Voiceye is the first application to integrate these different methods of input into a coding environment and to investigate the potential of this approach to support disabled coders. The paper also presented a novel approach to help address the slow typing issues associated with gaze using a common shorthand coding notation [26]. Participants provided positive feedback about Voiceye with physically impaired coders rating the application has having a good level of usability. Participant comments also identified key areas where future research will be important to help make the application and multimodal development approaches more accessible for disabled coders.

ACKNOWLEDGEMENTS

We are grateful to Microsoft for supporting this project through an “AI for Accessibility” grant. We also thank London RSI Support Center for supporting the recruitment of physically impaired developers.

REFERENCES

[1] Advanced Voice-Control, speech to code, program by voice, stop RSI. Retrieved April 11, 2019 from http://voicecode.io

[2] Atom. Retrieved January 29, 2020 from https://atom.io

[3] Mubbashir Ayub and Muhammad Asjad Saleem. 2012. A speech recognition based approach for development in C++. *IJCSNS* 12, 10: 110-114

[4] Imran Bajwa, Waqar Aslam Sarwar, and Irfan Hyder Syed. 2006. Speech Language Engineering System for Automatic Generation of Web based User Forms. In *Proceedings of the International Conference on Man-Machine Systems (ICOMMS 2006)*

[5] Aaron Bangor, Philip Kortum, and James Miller. 2009. Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of Usability Studies* 4, 3: 114-123.

[6] Richard Bates and Istance Howell. 2002. Zooming interfaces!: enhancing the performance of eye controlled pointing devices. In *Proceedings of the fifth international ACM conference on Assistive technologies*, 119-126.

[7] Tanya René Beelders. 2011. *Enhancing the user experience for a word processor application through vision and voice.* Ph.D Dissertation. University of the Free State.

[8] Tanya René Beelders and Pieter J. Blignaut. 2010. Using vision and voice to create a multimodal interface for Microsoft Word 2007. In *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications*, 173-176.

[9] Andrew Begel and Susan L.Graham. 2006. An assessment of a speech-based programming environment. In *Proceedings IEEE Symposium on Visual Languages and Human-Centric Computing*, 116–120.

[10] Andrew Begel and Susan L.Graham. 2005. Spoken programs. In *Proceedings IEEE Symposium on Visual Languages and Human-Centric Computing*, 99–106.

[11] Andrew Begel. 2005. Programming by voice: A domain-specific application of speech recognition. In *AVIOS speech technology symposium–SpeechTek West.*

[12] Brackets - A modern, open source code editor that understands web design. Retrieved January 29, 2020 from http://brackets.io

[13] Teresa Busjahn, Bednarik Roman, Andrew Begel, Martha Crosby, James H. Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm. 2015. Eye Movements in Code Reading: Relaxing the Linear Order. In *2015 IEEE 23rd International Conference on Program Comprehension*, 255–265.

[14] Emiliano Castellina, Fulvio Corno, and Paolo Pellegrino. 2008. Integrated speech and gaze control for realistic desktop environments. In *Proceedings of the 2008 symposium on Eye tracking research & applications*, 79-82.

[15] Central London RSI Support Group - Home| Facebook. Retrieved on January 30, 2020 from *https://www.facebook.com/CentralLondonRsiSupportGroup*

[16] Hashmeet Chadha, Satyam Mhatre, Unnati Ganatra, and Sujata Pathak. 2018. HTML Voice. In *2018 Fourth International Conference on Computing Communication Control and Automation*, 1–4.

[17] Tuhin Chakraborty, Sayan Sarcar, and Debasis Samanta. 2014. Design and evaluation of a dwell-free eye typing technique. In *Proceedings of the extended abstracts of the 32nd annual ACM conference on Human factors in computing systems*, 1573–1578.

[18] CodeMirror. Retrieved January 30, 2020 from https://codemirror.net

[19] Chris Creed. 2018. Assistive technology for disabled visual artists: exploring the impact of digital technologies on artistic practice. *Disability and Society* 33, 7: 1103-1119.

[20] Chris Creed, Maite Frutos-Pascual, and Ian Williams. 2020. Multimodal Gaze Interaction for Creative Design. In *Proceedings of the ACM conference on Human factors in computing systems (CHI).*

[21] Chris Creed. 2016. Eye Gaze Interaction for Supporting Creative Work with Disabled Artists. In *Proceedings of the 30th International BCS Human Computer Interaction Conference,* 1-3.

[22] Alain Desilets. 2001. VoiceGrip: A tool for programming-by-voice. In *International Journal of Speech Technology*, 103–116.

[23] Alain Désilets, David C. Fox, and Stuart Norton. 2006. VoiceCode: An Innovative Speech Interface for Programming-by-voice. In *CHI '06 extended abstracts on Human factors in computing systems*, 239–242.

[24] Electron - Build cross platform desktop apps with JavaScript, HTML, and CSS. Retrieved January 29, 2020 from https://www.electronjs.org

[25] Monika Elepfandt and Martin Grund. 2012. Move it there, or not?: The design of voice commands for gaze with speech. In *Proceedings of the 4th Workshop on Eye Gaze in Intelligent Human Machine Interaction*,1-3.

[26] Emmet — the essential toolkit for web-developers. Retrieved September 13, 2019 from https://docs.emmet.io/

[27] Hartmut Glücker, Felix Raab, Florian Echtler, and Christian Wolff. 2014. EyeDE: Gaze-enhanced software development environments. In *CHI'14*

Extended Abstracts on Human Factors in Computing Systems, 1555–1560.

[28] Benjamin M. Gordon and George F. Luger. 2012. English for spoken programming. In *6th International Conference on Soft Computing and Intelligent Systems, and 13th International Symposium on Advanced Intelligence Systems*, 16–20.

[29] Benjamin M. Gordon and George F. Luger. 2012. Progress in Spoken Programming. In *8th Student Conference*, 1–5.

[30] Dilek Hakkani-Tür, Malcolm Slaney, Asli Celikyilmaz, and Larry Heck. 2014. Eye gaze for spoken language understanding in multi-modal conversational interactions. In *Proceedings of the 2014 International Conference on Multimodal Interaction*, 263–266.

[31] Henna Heikkilä. 2013. EyeSketch: A drawing application for gaze control. In *Proceedings of the 2013 Conference on Eye Tracking South Africa*, 71–74.

[32] Anthony J. Hornof and Anna Cavender. 2005. EyeDraw. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 161-170.

[33] Thomas J. Hubbell, David D. Langan, and Thomas F. Hain. 2006. A voice-activated syntax-directed editor for manually disabled programmers. In *Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility*, 205-212.

[34] Robert Jacob and Keith Karn. 2003. Eye tracking in human-computer interaction and usability research: Ready to deliver the promises. In *The Mind's Eye*, 573–605.

[35] Yvonne Kammerer, Katharina Scheiter, and Wolfgang Beinhauer. 2008. Looking my way through the menu. In *Proceedings of the 2008 Symposium on Eye Tracking Research & Applications* , 213-220.

[36] Jan van der Kamp and Veronica Sundstedt. 2011. Gaze and voice controlled drawing. In *Proceedings of the 1st Conference on Novel Gaze-Controlled Applications*, 1-8.

[37] Per Ola Kristensson and Keith Vertanen. 2012. The potential of dwell-free eye-typing for fast assistive gaze communication. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, 241-244.

[38] Chandan Kumar, Raphael Menges, Daniel Müller, and Steffen Staab. 2017. Chromium based framework to include gaze interaction in web browser. In *Proceedings of the 26th International World Wide Web Conference*, 219–223.

[39] Thomas D. LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining Mental Models: A Study of Developer Work Habits. In *Proceedings of the 28th international conference on Software engineering*, 492-501.

[40] Päivi Majaranta, Ulla-Kaija Ahola, and Oleg Špakov. 2009. Fast gaze typing with an adjustable dwell time. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 357-360.

[41] Rinor S. Maloku and Besart Xh Pllana. 2016. HyperCode: Voice aided programming. *IFAC-PapersOnLine*, 263–268.

[42] Darius Miniotas, Oleg Špakov, Ivan Tugoy, and I. Scott MacKenzie. 2006. Speech-augmented eye gaze interaction with small closely spaced targets. In *Proceedings of the 2006 Symposium on Eye Tracking Research & Applications*, 67-72.

[43] Sahil Modak, Sagar Vikmani, Suril Shah, and Lakshmi Kurup. 2016. Voice driven dynamic generation of webpages. In *2016 International Conference on Computing Communication Control and automation*, 1–4.

[44] Martez E. Mott, Shane Williams, Jacob O. Wobbrock, and Meredith Ringel Morris. 2017. Improving dwell-based gaze typing with dynamic, cascading dwell times. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2558–2570.

[45] Thomas H. Park, Ankur Saxena, Swathi Jagannath, Susan Wiedenbeck, and Andrea Forte. 2013. Towards a Taxonomy of Errors in HTML and CSS. In *Proceedings of the ninth annual international ACM conference on International computing education research*, 75–82.

[46] Rakesh Patel and Mili Patel. 2014. *Hands free JAVA (Through Speech Recognition)*.

[47] Marco Porta and Alessia Ravelli. 2009. WeyeB, an eye-controlled web browser for hands-free navigation. In *2009 2nd Conference on Human System Interactions*, 210–215.

[48] David Price, Ellen Rilofff, Joseph Zachary, and Brandon Harvey. 2004. NaturalJava. In *Proceedings of the 5th International Conference on Intelligent User Interfaces*, 207–211.

[49] Tavis Rudd. 2013. Using Python to Code by Voice. Retrieved September 17, 2019 from https://pyvideo.org/pycon-us-2013/using-python-to-code-by-voice.html

[50] Stevche Radevski, Hideaki Hata, and Kenichi Matsumoto. 2016. EyeNav. In *Proceedings of the 9th Nordic Conference on Human-Computer Interaction*, 1–4.

[51] Jean K. Rodriguez-Cartagena, Andrea C. Claudio-Palacios, Natalia Pacheco-Tallaj, Valerie Santiago González, and Patricia Ordonez-Franco. 2016. The

Implementation of a Vocabulary and Grammar for an Open-Source Speech-Recognition Programming Platform. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility*, 447–448.

[52] Lucas Rosenblatt, Patrick Carrington, Kotaro Hara, and Jeffrey P. Bigham. 2018. Vocal Programming for People with Upper-Body Motor Impairments. In *Proceedings of the Internet of Accessible Things*, 1-10.

[53] David Rozado, Alexander McNeill, and Daniel Mazur. 2016. VoxVisio–Combining Gaze and Speech for Accessible HCI. *Resna 2016*.

[54] Susana Rubio, Eva Díaz, Jesús Martín, and José M. Puente. 2004. Evaluation of Subjective Mental Workload: A Comparison of SWAT, NASA-TLX, and Workload Profile Methods. *Applied Psychology* 53, 1: 61–86.

[55] Korok Sengupta, Min Ke, Raphael Menges, Chandan Kumar, and Steffen Staab. 2018. Hands-free web browsing. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications*, 1–3.

[56] Asma Shakil, Christof Lutteroth, and Gerald Weber. 2019. CodeGazer: Making Code Navigation Easy and Natural with Gaze Input. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1-12.

[57] Henrik Skovsgaard, Julio C. Mateo, John M. Flach, and John Paulin Hansen. 2010. Small-target selection with gaze alone. In *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications*, 145–148.

[58] Lindsey Snell and Mr Jim Cunningham. 2000.An investigation into programming by voice and development of a toolkit for writing voice-controlled applications. M. Eng. Report, Imperial College of Science, Technology and Medicine, London.

[59] Speech to Text API | Microsoft Azure. Retrieved January 30, 2020 from https://azure.microsoft.com/en-gb/services/cognitive-services/speech-to-text

[60] Talon 0.0.7.7 documentation. retrieved April 19, 2019 from *https://talonvoice.com/docs/index.html#document-index*

[61] The Eye Tribe. Retrieved January 29, 2020 from https://theeyetribe.com/theeyetribe.com/about/index.html

[62] Visual Studio Code - Code Editing. Redefined. Retrieved January 29, 2020 from https://code.visualstudio.com.

[63] Tobii Gaming | Eye Tracker 4C for PC Gaming. Buy Now at €169. Retrieved January 29, 2020 from https://gaming.tobii.com/tobii-eye-tracker-4c

[64] Amber Wagner and Jeff Gray. 2015. An Empirical Evaluation of a Vocal User Interface for Programming by Voice. In *International Journal of Information Technologies and Systems Approach*, 47–63.

[65] Qiaohui Zhang, Atsumi Imamiya, X. Mao, and K. Go. 2004. A gaze and speech multimodal interface. In *24th International Conference on Distributed Computing Systems Workshops, 2004. Proceedings*, 208–213.