

FloRead

Rohit Jha, Abhinav Mishra, Yoshiki Vázquez-Baeza, Charles Cruz, Robert Kronebusch, Jeremy Leu

Humans interact with digital screens on a daily basis, our computers, phones, e-readers and other appliances rely on our ability to read and understand the content that is being displayed. Although digital display technologies have seen remarkable innovations over the past few years, the way we consume this content has not changed too much for the past few decades. In this document we introduce FloRead, a system that aims to change the way in which we read documents, by leveraging eye-tracking sensor and luminosity sensing. Using the eye gaze, it helps the reader focus in the section of the screen where the attention is currently centered at. Additionally, with the luminosity information, it modifies the contrast of the text and the background on the screen.

CCS Concepts: •**Human-centered computing** → *Ubiquitous and mobile computing systems and tools*;

Additional Key Words and Phrases: Eye-tracking

ACM Reference Format:

Charles Cruz, Rohit Jha, Robert Kronebusch, Jeremy Leu, Abhinav Mishra and Yoshiki Vázquez-Baeza. 2016. FloRead. *ACM Trans. Appl. Percept.* -, -, Article - (December 2016), 15 pages.

DOI: 0000001.0000001

1. INTRODUCTION

Despite continued advances in technology, the reading experience has still been dominated with the use of physical (printed) texts. Even though ebooks and printed text usage has increased in the 21st century, printed mediums have still dominated the education and research spaces. The differences in space and weight may be appealing, yet printed texts do not carry the problems of eye strains and content retention [Kraft 2015].

Unlike the simplicity of paper and text, a lot more information is present on the computer screen. There's the window that contains the reading content, the taskbar or dock of current running applications, and if the reading content is from a website, a few ads and website-related information. All these factor into the problem of information overload the occurrence in which information present surpasses our ability to process the information.

We present FloRead, an eye-tracking enabled reading experience that we designed as a research project for the CSE 118/218 Ubiquitous Computing course. The goal of FloRead is to provide readers of online articles and documents a natural interface for reading. We attempt to solve the problem of information overload by filtering out the main reading content from the original, cluttered webview. In order to do so, we have combined eye tracking and light sensing - a Tobii EyeX eye tracker provides us the means to determine the text currently being read, and a photosensor allows us to adjust the visual contrast of the article or document according to the ambient light.

The demo website for FloRead can be accessed at: <http://jeremyleu.com/floread/>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2016 ACM. 1544-3558/2016/12-ART: \$15.00

DOI: 0000001.0000001

The outline of this report is as follows: in section 2, we explain the motivation behind developing FloRead and relevant background details. In section 3, we will discuss the design of our system, including initial prototypes we had planned and its final conception. In section 4, we explain the system architecture of FloRead and its three main modules—light-sensing, eye-tracking, and the processing units. In section 5, we review our testing and evaluation of the system we created over the quarter. In section 6, we explain how our team collaborated for working on this project, the issues we faced while doing so and how we handled them. Finally, we review our work and mention possible future research directions for FloRead.

2. MOTIVATION AND BACKGROUND

The continued development of technology has allowed the efficiency of expressing physical text and graphics into an electronic form. Computers have given people the convenience of accessing, archiving, and organizing their data with ease. Tablets and e-readers have allowed people to read various e-books and e-magazines without the size and durability limitations that come with their physical counterparts. Electronic texts can easily be purchased online without the need of traveling to specific locations. To this degree, electronic texts have become more mainstream than the physical copy.

The widespread adoption of personal computers and tablets have contributed to the popularity of electronic texts. These electronic devices are lightweight and portable in comparison to a set of books. These devices allow the user more customization than a book such as increasing text size or the size of the canvas of text by adjusting the size of a window. Yet, there are still some disadvantages of electronic books than their physical equivalent.

Over the years, the development and richness of graphical user interfaces and software has increased. However the human-computer interaction has been limited by traditional input devices. Using a keyboard or mouse to read and navigate through content lowers the immersion that the user experiences with the computer. When applied to reading, scrolling and clicking through the pages of text displayed on a screen is not as ubiquitous as the physical touch of turning a page in a book. In addition, books do not emit light, unlike LCD or LED screens. Furthermore, continued reading in different bright or dark environments can contribute to eye strain.

Today, with the abundance of sensors and their cheap costs, advancing the electronic reading experience is possible. With eye-tracking devices, control of computer screens can be off-loaded from the traditional mouse and keyboard to the human perceptual organ—the eye. Furthermore, reducing the stress of the eye on the LCD screen can be averted through automatic brightness adjustments from a light sensor.

Our solution focuses on providing an enhanced reading environment, enabled by leveraging eye-tracking and ambient-light sensing. In recent years, quotidian use of light-emitting devices like computers, phones and tablets, has become more pervasive in our society. Professionals and students both rely on these systems to perform everyday activities like reading. As shown in 2015 [Kraft 2015], unaltered usage of these devices before sleep, can noticeably impact the mood of a user and affect sleep.

Empirically, we have observed that scholars and students performing research, where electronic reading materials are essential, often prefer to print out the materials than to consult them using their computers, due to their machines constantly triggering distractions that degrade the reading experience, or requiring interactions that don't exist in the paper space (such as modally scrolling). We believe using an eye-tracking-enabled system would help the machine guide their attention, adjust the screen contrast and assist them when performing tasks such as looking up a word on the dictionary, or consulting a figure or table located at some other place in the same document. Altogether ameliorating some of the most prominent annoyances that reading electronic documents present.

Attention aware systems: Theories, applications, and research agenda

Claudia Rodia ^{*,†}, Julie Thomas ^{*}

^{*} Computer Science Department, The American University of Paris, 31 Avenue Bruguier, 75007 Paris, France

[†] International Communication Department, The American University of Paris, 31 Avenue Bruguier, 75007 Paris, France

Available online 30 January 2016

Abstract

Human perceptual and cognitive abilities are limited resources. Attention is the mechanism used to allocate such resources in the most effective way. Current technologies, in addition to allowing fast

access to information and people, should be designed to support human attentional processes in which they improve further study. This paper analyzes the issues related to the design of systems capable of such support: attention aware systems. We introduce the research aimed at understanding and modelling human attentional processes, including perceptual and cognitive processes as studied in cognitive psychology, as well as theoretical, scientific, and social aspects related to attentional mechanisms. We use four current approaches to the design of attention aware systems along three major features: detection

Fig. 1. Initial prototype of how we envisioned FloRead to modify a document to enhance its readability.

3. DESIGN

The final product that we created, ended up being remarkably different from our original design (see Figure 1), most notably, we changed the scope of the focus from being at the operating system-level to being only restricted to one application. In the rest of this section, we explain (in chronological order), the events that occurred throughout the development of FloRead, and their consequences.

Although, eye tracking can potentially be used to aid any application, the reality is that the developer tools are not quite that powerful yet, at this time the Tobii EyeX Software Development Kit (SDK) [AB 2014a] is only available on the Windows platform through Windows Forms (WinForms) or Windows Presentation Foundation (WPF). While this may not sound problematic, at a first glance, in retrospective this was one of the most transformative features. First, it required that we worked on Windows, even though nobody in the team uses Windows as their main operating system. Consequently, we did not have any experience developing for this platform, so getting our development environments up and running was cumbersome at first. Perhaps the most important aspect to note here is that this limited us to only being able to work within the context of a single application, so we changed our design to fit with this artificial requirement.

Settled inside a WinForm application, the next task was finding a way to display content in this context. We considered using a text box widget, using a PDF document browser, or to use a WebView widget [Microsoft 2016b]. We chose to use the web view, specially since there is already a native WebView class in the .NET Framework. An added benefit of this approach is the ability to display a variety

of different file types in a rather seamless way, however the biggest downside was having to maintain a hybrid codebase (between JavaScript and C#). With this transition, we also opted to move away from PDFs, and instead we centered our attention on reading online articles, by doing so we saw an opportunity to use a web parser ¹, which gave us more flexibility on the presentation of the data, we could now have total control over the text and its format.

In our proposal, we emphasized the ability to use eye gestures to enhance the reading experience, however as we worked through the weekly readings, we became aware of some challenges that only exist in the eye-tracking space, for example the *Midas touch* effect. Going forward, we decided to only use eye-tracking as an interface for the navigation of the document.

Moving forward, the next problem was dealing with Microsofts implementation of the web standard. As we found out, the blurring functionality in CSS and per-line formatting are not available in any of Microsofts web browsers (Internet Explorer nor in Microsofts Edge), but are in every other browser (Google Chrome, Mozilla Firefox and Safari). We considered that even if we were to find workarounds these missing features, it was likely that we would find another missing feature. Therefore we decided to see if we could use a different browser, this is when we found out about CEFSharp [Developers 2016], an implementation of the Chromium web browser that can be invoked from within a C# program.

As part of the project proposal, we requested the Intel Galileo board [Intel 2016], and Intels Internet of Things Kit (IoT Kit). However, when we tried to setup the board for development, we realized we were missing two additional components: 6 pin FTDI cable TTL-232R-3V3, and SD Card. We chose not to purchase these parts, and instead we bought an Arduino Uno board [Arduino 2016b], and a Mini Luminance Light Sensor (by Phantom YoYo). We specifically chose to use Arduino for its broad community, and extensive amount of educational resources (consequently integrating the light sensing with the C# application was straightforward).

While we originally wanted to use the light sensor to adjust the brightness of the screen, we further considered this idea and realized that for most modern computers this functionality would be redundant, and likely not useful. Therefore the luminosity is instead used to adjust the color scheme of the text. For example, in a room with reduced ambient light, the text is rendered in a light color and dark background, and as the light increases, the color of the text is reduced in lightness and the background is increased in lightness.

Finally, the design of the application centered around a webpage that implements a simple interface where the user can enter a URL, and the content of the page is extracted (only restricted to images and text). Most notably, all ads, and elements that may act as a distraction, are removed. The text is then formatted on a single page, that receives information from the eye-tracker and the luminosity sensor. With this information, the contrast of the content is automatically adjusted and the user can seamlessly navigate through the document. The overall timeline of the project can be seen in Figure 2.

4. SYSTEM DEVELOPMENT

4.1 Architecture

FloReads system architecture comprises broadly of three modules namely Light Sensing, Eye-tracking and Processing units. Figure 3 shows these three modules and their interactions with each other and the reader. We now explain each of these in detail.

4.1.1 Eye-tracking. Our eye-tracking unit consists of the Tobii EyeX eye tracker. The eye tracker contains a near-infrared micro-projectors, optical sensors and image processing. The eye tracker sends

¹In this context, we refer to a web parser as a service capable of extracting text and image content from a website. This parser will not include any information that is not central to the article being presented on that page.

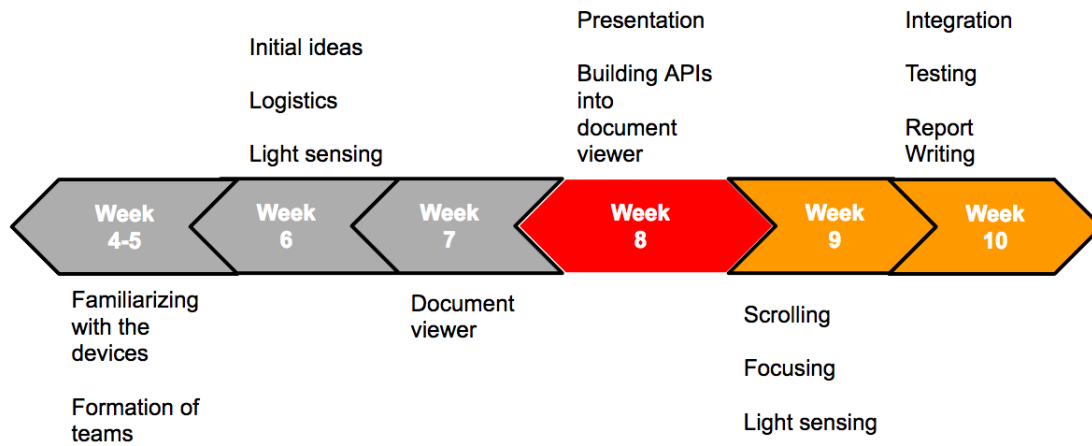


Fig. 2. Weekly timeline of the project, and the high-level tasks that we focused on.

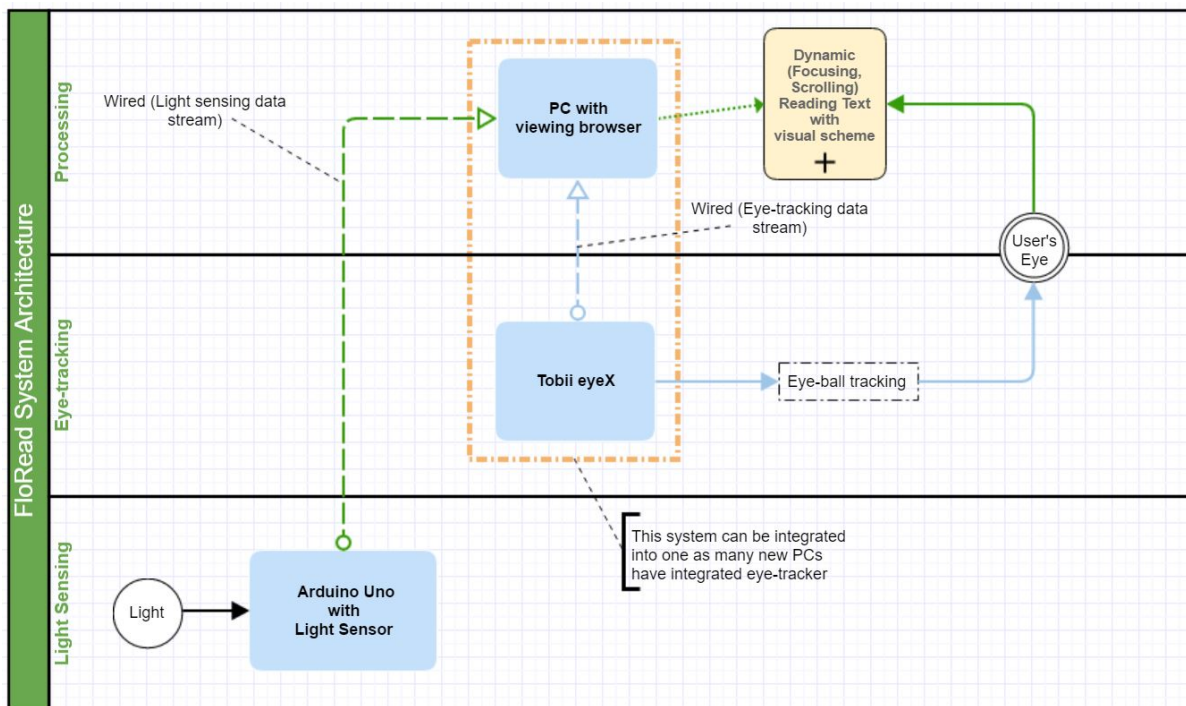


Fig. 3. System architecture for FloRead.

out infrared rays and the near-infrared micro-projectors creates a reflection pattern on the users eye. The image sensors present on the eye-tracker registers the image of the user, the users eyes, and the projection patterns, in real time (see Figure 4). Image processing unit within the eye-tracker is then used to find features of the user, the eyes and projection patterns. When a user looks at his/her screen the eye tracker records all these data calculating the right angle that the eye makes with the screen and the tracker. Various mathematical models are then used to calculate the eyes position and the gaze point. This gaze point is not fed into the processing unit which has its own CPU and a viewing browser.

4.1.2 Light Sensing. The light sensing unit consists of an Arduino Uno microcontroller along with a photosensor. The light from the users environment in which he/she is trying to read the text on the screen is sensed and fed into the processing unit. The light sensor present on Arduino detects current ambient light levels and helps setting the right brightness and visual scheme environment on the web browser.

4.1.3 Processing Unit. The processing unit comprises of a personal computer running Windows 10. The eye-tracking and Arduino are connected to it through a wired connection. The output from the processing unit is displayed on the screen where the user is performing his/her reading activity. The user first inputs the URL of the reading material he wants to read into the FloRead application. The URL is parsed and displayed on the screen. Meanwhile the eye tracker performs its own processing and sends the gaze point coordinates to the application. Parallely the Arduino does its work and sends the current light ambient level to the application. Using these two types of data the application performs its processing and ultimately modifies the reading content and its environment. Hence, giving out an enhanced reading experience to the user.

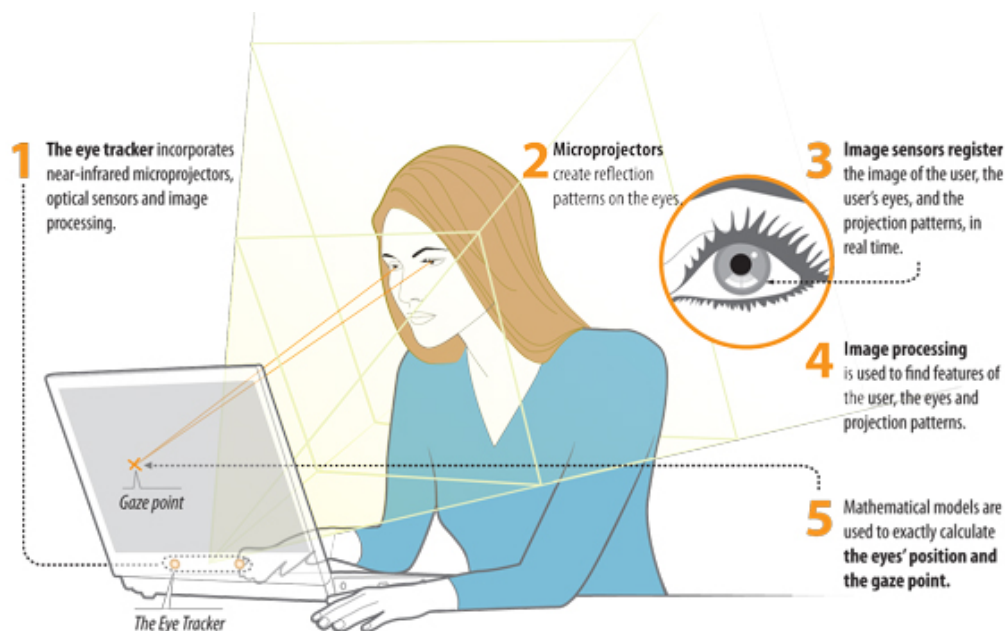


Fig. 4. Working of the Tobii EyeX eye tracker (source: Tobii EyeX documentation [AB 2014b])

4.2 Technologies used

4.2.1 Devices. Since our idea was based on tracking gazes and light sensing, we started with the Tobii EyeX eye tracker, Intels Galileo Board and Intels Internet of Things (IoT) kit. However, as mentioned previously, for a variety of benefits, we switched from the IoT kit to using an Arduino Uno microcontroller and the Mini Luminance Light Sensor. In order to speed up the development process and eliminate any delays that may arise due to unavailability of devices, we checked out two Tobii EyeX eye trackers from the lab. In addition, we also bought another microcontroller and photosensor to further hasten development.

4.2.2 Development Environment. The Tobii EyeX SDK provides support for the .NET Framework with C# and C++ but since the majority of our team members were more comfortable with C#, we chose that as our programming language. Moreover, the EyeX eye tracker is currently only supported on Windows and therefore we used Windows 10 on our laptops (or installed it in a virtual machine) for development. As part of the development environment, all the team members installed Visual Studio 2015 Community Edition [Microsoft 2016a] as the IDE and the members working with Arduino Uno installed the Arduino Software IDE [Arduino 2016a] too and programmed the microcontroller using Arduinos C derived programming language. To handle the formatting of online documents, we used HTML and CSS, and relied on JavaScript for modifying the contents representation at runtime.

4.2.3 Services, Libraries and APIs. In order to convert an online article or document to a suitable format that is easily readable, we used the Mercury Web Parser by Postlight [LLC 2016], which is a web service that takes in a URL and returns the title and HTML content of the page as strings. We then displayed this HTML containing the articles content in a single column and also eliminated many embedded advertisements and distractions. We used the Lining.js [<http://zencode.in/lining.js/> 2016] JavaScript library so that we could apply filters on or remove filters from one line of text, thereby allowing us to add focusing styles to the line currently being read. In order to easily perform this, we used the jQuery [jQuery 2016] library. Since the built-in WebView from the .NET Framework was insufficient for us, we used the CEFSharp (Chromium Embedded Framework for WinForms and WPF) library that supported a recent version of the Chromium browser engine for rendering web pages.

4.3 Features

4.3.1 Automatic Visual Scheme Adjustment. This feature allows a user to read online articles and documents without any strain to their eyes. We drew inspiration for this feature from some blue-light filtering software (e.g., f.lux, Twilight and RedShift) that change the hue of a screen to reduce blue colors and light frequencies. We believe that this crucial functionality will help readers in dim light, such as during nighttime, and improve their health by removing light frequencies that disrupt sleep.

Depending on the luminosity level, i.e. ambient light detected by our photosensor, we can change the online documents visual scheme to make sure that the readers eyes are not strained. Accordingly, we have created five visual schemes (or themes) that our document viewer automatically switches to, as can be seen in Figures 5 and 6.

4.3.2 Automatic Scrolling. Since reading most articles or documents on a computer require the reader to scroll down and up, our hypothesis was that adding functionality for automatically scrolling based on where the readers gaze is, will make for a better and natural reading experience.

We constantly obtain the readers current gaze point on the screen from the EyeX in the form of a GazePointDataStream and after every 499 gaze points we record the X and Y coordinates of a gaze point. Discarding the others results in better efficiency, a smoother experience since the rate of receiving gaze points from a GazePointDataStream is quite high.

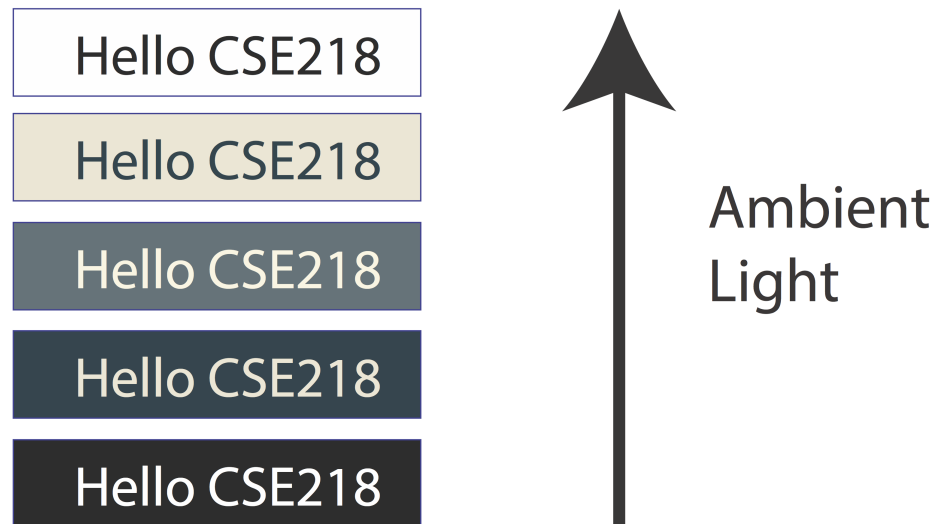


Fig. 5. Various visual schemes for displaying text in our application. The gradient corresponds to an increase in ambient light intensity from bottom to top - the scheme is white on black for dim lighting, and black on white in bright lighting.

Once we know where the reader is currently on the document, we can determine whether they are somewhere near the top or bottom of the current viewport. If they are reading somewhere around the top of a page, we scroll up a bit and if they are reading somewhere around the bottom, then we scroll down a bit, as shown in Figure 7.

4.3.3 Focusing. A common issue faced by readers of long and dense articles or documents is that focusing on a line of text may sometimes be difficult. To facilitate reading, our hypothesis is that if we can blur out all the text except the current line being read, and slightly enlarge this line, then it should make for a better reading experience.

To implement this feature, we again rely on the gaze points from EyeX and use the same mechanism mentioned under the automatic scrolling feature section. Once we know the X and Y coordinates, we move the mouse cursor to the gaze point but hide the cursor. The Lining.js JavaScript library allows us to format each line of a page, so we simply blur the entire page using a CSS filter and remove the blur filter for the line currently under the cursor, which is the line currently being read, as shown in Figure 8.

5. TESTING AND EVALUATION

We tested all aspects of the application thoroughly. Even in early stages of development, we performed A/B tests on number of different effects to emphasize the text that the readers eyes were focused on. We knew that we wanted to make reading on the computer a less distracting, more natural-feeling experience, but we had different ideas about how we would implement these ideas. Additionally, we



Fig. 6. Screenshots of the document viewer in each of the five schemes shows the readability of the text is enhanced.



Fig. 7. Gazing on the bottom 10% of the screen scrolls the document down. Similarly, gazing at the top 10% of the screen will cause scrolling up.

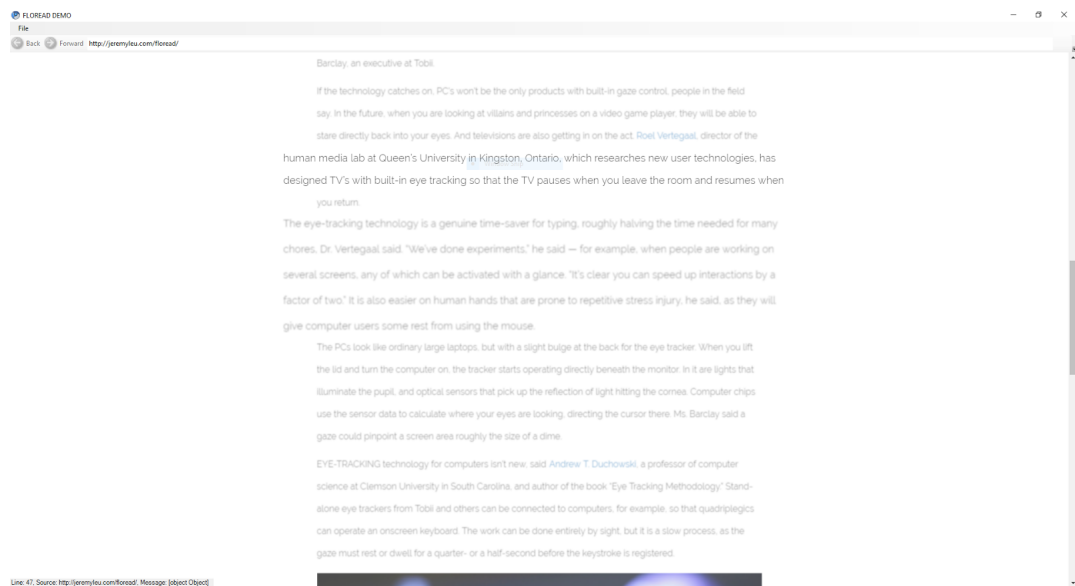
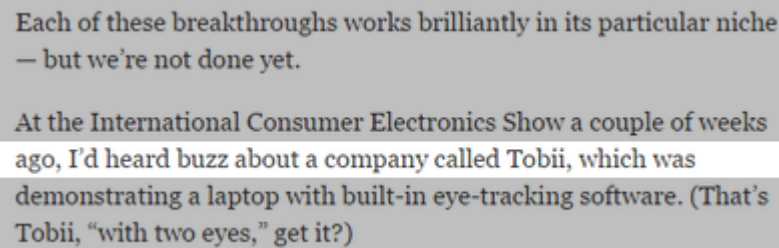


Fig. 8. Focusing causes the text being read to be magnified by 10% and the remaining document is blurred.



Each of these breakthroughs works brilliantly in its particular niche
— but we're not done yet.

At the International Consumer Electronics Show a couple of weeks
ago, I'd heard buzz about a company called Tobii, which was
demonstrating a laptop with built-in eye-tracking software. (That's
Tobii, "with two eyes," get it?)

Fig. 9. Early mock-up of our *blurring* system.

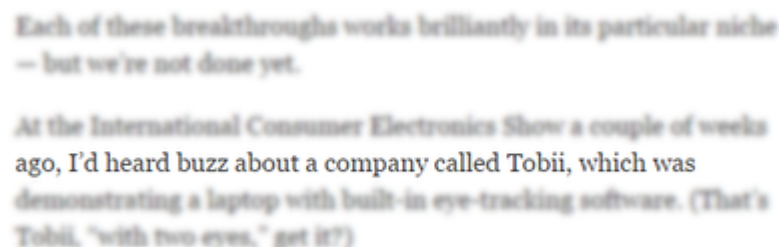
knew that we would have automated scrolling, an aspect we had to take into consideration when deciding the implementation of the effect.

One of our initial ideas was to display a blinds effect that simply darkened the area around the text that the readers eyes were not currently focused on (see Figure 9 and 10). We considered this option because it would easily bring out the line that the reader was currently looking at, and seemed fairly simple to implement, as it would be simply a semi-transparent layer.

The alternative we thought about for the A/B test was a blurring of the surrounding text, as shown in 9. Proponents of this alternative thought that it felt more natural for text to fade out from view and found the blurring less obtrusive and more a more natural effect. However, it seemed more difficult to implement, as the only blurring technologies we could find with WinForms applications (the format we had already selected at this point in development) would rasterize the text and replace it with an image, making it no longer selectable.

Eventually, the feedback received was decisive: users agreed with the blurring effect feeling more natural, especially as the document scrolled by itself. We overcame the obstacle of preserving the texts integrity by displaying the text as HTML, and using a CSS filter to blur the text, a technology that does preserve the text and allows it to be selectable.

After having decided on the blurring, we created prototypes for user testing and asked for their open-ended feedback, which we used to make design decisions about the application. Design decisions that were results of the user testing to improve the user experience included increasing the spacing between each of the lines to make it easier to read, and also focusing on the line immediately following. We decided to remove the blurring effect on not only the line that the eye tracker determined that the reader was reading, but also the following line, in part to alleviate a problem often mentioned in user feedback: that there was significant lag whenever the user changed lines. We also worked to minimize



Each of these breakthroughs works brilliantly in its particular niche
— but we're not done yet.

At the International Consumer Electronics Show a couple of weeks
ago, I'd heard buzz about a company called Tobii, which was
demonstrating a laptop with built-in eye-tracking software. (That's
Tobii, "with two eyes," get it?)

Fig. 10. Final implementation of FloRead.

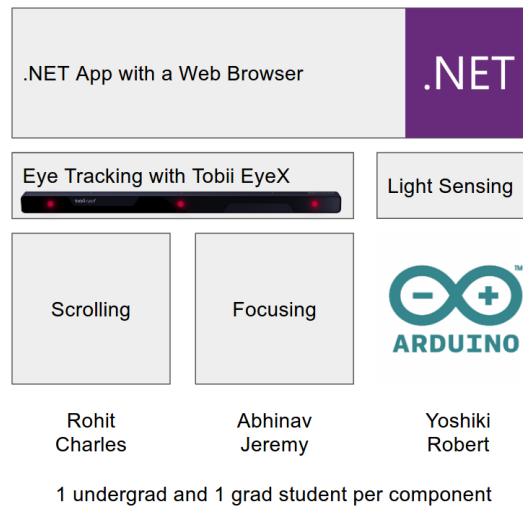


Fig. 11. Early mock-up of our *blurring* system.

this lag and optimize the application, to a point that we felt like the application was usable without the reader becoming frustrated by having to read a blurred line.

Finally, the last change we made based on user feedback was to increase the font size of the line that was focused on. User feedback to this change was positive: the magnification effect further emphasized the line and relieved some stresses usually found when reading from a computer screen. For the automatic visual scheme adjustment, we finalized the five schemes after a usability test on a set of sample articles that are shown in Figures 5 and 6.

User feedback with the current version was much more positive than our initial prototypes: the major problems were eased and many found it a viable alternative to printing out an article and reading the hard copy instead of using the computer to read.

6. COLLABORATION

6.1 Team Structure

Before we were able to form sub teams we had decided to break the project up into the three key aspects of our FloRead - Light Sensing, Scrolling, and Focusing. Each of these sub-teams were made up of one undergraduate and one graduate student. The three sub-teams were as follows:

—**Light Sensing (Arduino):** Yoshiki Vazquez Baeza (218) and Robert Kronebusch (118).

—**Scrolling (EyeX):** Rohit Jha (218) and Charles Cruz (118).

—**Focusing (EyeX):** Abhinav Mishra (218) and Jeremy Leu (118).

The two key factors for selecting these sub-teams were available free times and areas of interests. To figure out what time each person would be available we conducted an online survey that was able to let us know each others schedule. This step alone may have been the single most important factor in the success of our teams. To discover what portion of the project everyone wanted to work on we arranged a meeting and asked everyone what they wanted to do. Luckily, for us, there was no disagreement about who would do what.

For the project report our CSE 218 members decided that we would split up sections for undergraduate and graduate students. The work would be equally divided and each team member would also be

able to pick what section they would like to write about. The following is a breakdown of what team member was responsible for each section of our project report:

- (1) **Executive Summary:** Yoshiki Vazquez Baeza (218) and Rohit Jha (218).
- (2) **Introduction:** Charles Cruz (118).
- (3) **Motivation and Background:** Charles Cruz (118).
- (4) **Design:** Yoshiki Vazquez (218).
- (5) **System Development:** Rohit Jha (218) and Abhinav Mishra (218).
- (6) **Testing and Evaluation:** Jeremy Leu (118).
- (7) **Collaboration:** Robert Kronebusch (118).
- (8) **Conclusion:** Yoshiki Vazquez (218).

In order to merge each section into one cohesive working document we decided to collaborate on Google docs before finalizing our report. Google Docs allowed us to easily read and edit the entire document before we converted it into a LaTeX document for our final report.

6.2 Collaborations Between CSE 118 and CSE 218 Members

Our main challenge for collaboration was determining convenient times for all members to meet together. However, after we conducted a survey as mentioned in the previous section we no longer had any time conflict issues.

For the duration of our project our main form of communication was Slack. If you are unfamiliar, Slack is an online chat tool that allows users to easily share files and messages between users. Slack was key for all of us to keep up with meeting times and also quickly discuss any sort of issues that has occurred. Although Slack was good for minor issues we found that the best way to solve larger problems would be to simply meet up as a team and resolve the issue in person.

Another tool we used to collaborate was ZenHub. None of us had ever used ZenHub before, but we found it easy to understand the big picture of the project as well as communicate key milestones amongst other subteams. Because ZenHub is an extension over GitHub we were also easily able to see how far each of the other sub teams have achieved within their individual sub groups.

Overall there was little to no disparity between who was an undergraduate and who was a graduate student. During our meetings, every member of the team was heard and all input was considered amongst the team.

6.3 Problems and Resolutions

Throughout the development of FloRead we had three problems that we would consider major. We tackled the first two of these by using a software development methodology called mob programming, which is an extension of pair programming, wherein a large group (in our case 5 or 6) of developers work on a single computer.

First, we had to decide if we were going to have to develop our own web viewer in C# or use an existing framework that would allow us to do what we wanted. To solve this problem we had a miniature hack-a-thon for about four hours to come up with a solution as a group. The result of our hack-a-thon was that we would be better off using our own custom web view. Not only did a hack-a-thon give us a solution, but it also gave us a better understanding of how we would work together as a team.

The second major problem was a result of our choosing our web viewer. We quickly learned that the web viewer we were using was rendering HTML with the Internet Explore/Edge engine. We quickly discovered that the blurring effect we were planning to use was not supported with the Edge engine. Once, again we met as a team to decide how to tackle this issue. We had explored the option of using

CEFSharp as our browser engine. Unlike the default C# WebView, CEFSharp uses the Chromium engine. By using Chromium, we were then able to add the blurring effect that we had desired.

Although we were all excited to work on a platform that was new to everyone (nobody of us had developed software for Windows before), we found this operating system to be harder to develop for than UNIX-based systems. We all experienced problems in one way or another, from finding a way to use and install Git (Visual Studio has git integration, but is not as flexible as the command line), to realizing that Microsoft's implementation of the web standard is rather different compared to other browsers. A simple solution, would have been to move away from Windows, but given that the Tobii Eye-x SDK is only available for this OS, we did not have a choice and we had to relearn many of the things that we already knew how to do in Linux and macOS.

Our last major problem was not being able to use the IoT kit provided. For some reason, the kit did not include all the necessary equipment. We held a meeting and decided that using a basic Arduino and a light sensor would be the best approach.

Over the course of our work we also faced many small and big technical problems. One such problem was the integration of our Tobii EyeX related code with our viewer. We faced issues like not being able to get the gaze point coordinate in the web-viewer part of our project. This was a critical problem. We solved this by looking at many examples from the sample minimal apps that Tobii SDK provides.

Another issue was that our focusing module was rendering and taking the position of gaze point instantaneously and that caused a lot of disturbance when it came to usability. We solved this by keeping a timer in the code for getting the user's gaze point as a user generally spends some time looking/reading at a line before moving one to the next one.

All other problems that occurred were generally handled within the subteams.

7. CONCLUSION AND FUTURE WORK

In this project, we explored the idea of creating an eye-tracking enhanced reading experience. To reduce the readers' eye strain, we integrated light sensing to adjust the contrast of the text presented on screen (varying the color of the text and the color of the background). Although we found a number of technical issues in different areas of the development of this system, we also found workarounds for these problems, and came to realize a system not too far from our original design. In many parts of this project, we were largely benefited from the availability and documentation of several open source software packages, without we would have likely not met our goals.

As the technology for eye-tracking continues to develop, more applications will emerge, where eye-tracking is not the means to an end, it instead will act as a supplement to regular user interactions. Web browsers, and text editors are likely to have the most promising opportunities. However, in order for this to be possible, there are still several obstacles that need to be surpassed.

For now, we consider, both users and developers will suffer from the shortcomings of these new technologies. On the one hand, users are still required to purchase an additional piece of hardware that needs to be mounted on a monitor (or more inconveniently carried around together with a laptop), considering that the applications that integrate eye-tracking are still pretty limited and mostly novelty tech demos, this is not be a good investment for most computer owners. On the other hand, developers are tasked to integrate these devices in an ecosystem that is not designed to handle input from anything other than a mouse and keyboard. We were specially affected by this, although our initial intent was to be able to seamlessly integrate with the operating system, we quickly realized that the SDK imposed certain limitations that greatly restricted the potential of our project.

Moving forward, we believe there are two main areas of opportunity that could be refined and explored. First the ability to integrate the eye-gaze as part of existing software, and second integrating the ability to create annotations based on the users focus.

Regarding integration, an application like FloRead would need to become a software extension, this should not be a 3rd party program that you have to start before you can read something. It should be as easy as clicking a button from within your browser or text editor. For example, scientific journals that offer HTML versions of their articles could integrate with this browser extension, such when you read Figure 11, the extension would recognize this and display the figure and its legend. Many of these journals already offer alternative readers, but these are cumbersome at best. Though this has to be done with caution and used sparingly, if the user is faced with too many distractions, their attention might be affected, and this would likely result in the user losing their point of focus (this would ultimately defeat the purpose of a system like FloRead).

As for annotations, we think that a proper analysis of the attention that we devote to certain regions of a text and the reaction we reflect, can reveal (for example) the difficulty that we have at understanding the meaning of a sentence. Using this information, a new system could automatically generate brief summaries, where these would seek to emphasize the points that were tougher, or that resulted in the strongest reaction from the reader. The main challenge here would be properly detecting the reactions of a user, and generalizing this to broader audience. As noted by one of the instructors in our proposal, using other information like the dilation of the pupil, can inform us when a reader is in a deep state of concentration.

The current implementation of the system, gave us an opportunity to experience what the future may hold ahead of us, both as users and as developers. In a near future, peripherals like this, or the Leap Motion, will likely already be part of a computer, and will not require that additional hardware be purchased separately. In the same way that web cameras saw a transition from being (almost exclusively) external devices that would mount at the top of a monitor, to now being hardware that can be found even in the most affordable personal computers, many of these technologies will begin to become more quotidian. With a base-environment like this, developers will be motivated to create programs and interactions that take advantage of the technology.

REFERENCES

- Tobii Technology AB. 2014a. The Tobii EyeX SDK for .NET. (2014). Retrieved December 4, 2016 from <http://developer.tobii.com/eyex-sdk/dotnet/>
- Tobii Technology AB. 2014b. What is eye tracking? (2014). Retrieved December 4, 2016 from <http://developer.tobii.com/what-is-eye-tracking/>
- Arduino. 2016a. Arduino Software. (2016). Retrieved December 4, 2016 from <https://www.arduino.cc/en/Main/Software>
- Arduino. 2016b. Arduino UNO Board. (2016). Retrieved December 4, 2016 from <https://www.arduino.cc/en/Main/ArduinoBoardUno>
- CEFSharp Developers. 2016. CEFSharp. (2016). Retrieved December 4, 2016 from <https://cefsharp.github.io/>
- <http://zencode.in/lining.js/>. 2016. lining.js. (2016). Retrieved December 4, 2016 from <http://zencode.in/lining.js/>
- Intel. 2016. Intel Galileo Board. (2016). Retrieved December 4, 2016 from <https://software.intel.com/en-us/iot/hardware/galileo>
- jQuery. 2016. jQuery. (2016). Retrieved December 4, 2016 from <https://jquery.com/>
- Amy Kraft. 2015. Books vs. e-books: The science behind the best way to read. (Dec. 2015). Retrieved December 4, 2016 from <http://www.cbsnews.com/news/kindle-nook-e-reader-books-the-best-way-to-read/>
- Postlight Labs LLC. 2016. Mercury: Make any webpage make sense. (2016). Retrieved December 4, 2016 from <https://mercury.postlight.com/web-parser/>
- Microsoft. 2016a. Visual Studio Community. (2016). Retrieved December 4, 2016 from <https://www.visualstudio.com/vs/community/>
- Microsoft. 2016b. WebView Class Documentation. (2016). Retrieved December 4, 2016 from <https://msdn.microsoft.com/library/windows/apps/windows.ui.xaml.controls.webview.aspx>

Received December 2016; revised December 2016; accepted December 2016