FlightCrew Browser: A Safe Browser for Drivers

by
Andrés Humberto López-Pineda
S.B., Massachusetts Institute of Technology, 2012

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

June 2013

FlightCrew Browser: A Safe Browser for Drivers

by
Andrés Humberto López-Pineda
S.B., Massachusetts Institute of Technology, 2012

Submitted to the Department of Electrical Engineering and Computer Science
May 24th, 2013
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Drivers have information needs they want to solve while driving, but current mobile browser interfaces can bring forth safety issues when users browse the web even though their attention is required elsewhere, as it is during driving. FlightCrew Browser is a crowd-adapted web browser using speech input, touch input, speech output, and visual output in appropriate, informative, and safe ways to empower the driver of a car to investigate an evolving information need. Our system uses human workers to do browsing interactions for the user, as well as pick data from webpages that will be returned to the user. We use three workers at a time in order to provide quality control by using a voting system to pick what answers the workers believed to be best for the user. The workers can hear the driver's query and see the last page the driver received answers from in order to provide a shared context.

FlightCrew Browser provides a low-risk way for users to access the web when they are commuting or traveling alone. We evaluated our system using metrics and tests similar to those used by the National Highway Traffic Safety Administration and found that it is safer for drivers to use than existing mobile browsers.

Thesis Supervisor: Robert C. Miller
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgements

I would like to thank my family and friends for their support and guidance while writing this thesis. In particular, my brother Bokil and my parents Enrique and Renee are the most supportive people I can imagine. I would also like to thank my advisor Rob Miller, and the rest of the UID group at CSAIL. I am grateful for the work Robin Cheng and Kyle Murray put into the FlightCrew platform. Special thanks to all of the group members that directly helped with my writing, including Katrina Panovich, Kyle Murray, Tom Lieber, Haoqi Zhang, Rob Miller, Haoqi Zhang, and Juho Kim. Thanks to everyone who supported me through my final year, including Crystal Noel, Blair Gagnon, and Taylor Ferry, among many others. I wouldn't be where I am today with all of my friends from Eagan, MN: Jordan Lyseng, David Engler, Courtney Nelson, Andrew Eull, Tommy Barker, and many more. A special thank you to the teachers in Pinewood, Dakota Hills Middle School, and Eagan High School for preparing me for my future. Finally, thank you to all of my friends in Kappa Sigma Gamma-Pi, especially for those who helped me as an MEng student, such as Dylan Hadfield-Menell and Ryan Schoen. Thanks to Ford for funding our project.

Chapter 1

Introduction

In this project, we focus on empowering users who are trying to actively drive a car and browse the web safely. In order to accomplish this, we developed FlightCrew Browser (FCB). This thesis outlines the FCB system, which is used by the driver through an Android application installed on a tablet placed on a car's dashboard. The driver uses speech and touch input to create queries and results are displayed on the screen and read aloud. Queries are answered by several crowd workers through Amazon's Mechanical Turk [1]. The Mechanical Turk platform is used by many crowd-powered applications and allows for the anonymous recruitment of humans for use in small, paid tasks. Using multiple workers allows us to provide quality control.

In this chapter, we provide motivation for this work, as well as describing our research contributions and scope. Finally, we give a brief overview of our system and the findings from our user studies. In the remaining chapters, we present the design and implementation for our interfaces, a user study evaluating each interface, and a discussion on future work in this space.

1.1 Motivation

Mobile users have informational needs, including trivia, directions, and point of interest data [18]. The same study found that 69% of surveyed users solved their needs by using the web, calling a proxy to find the information, or by directly calling the source of the information. However, making phone calls or using a mobile browser can be difficult while driving, because the actions a driver takes behind the wheel have a direct impact on the user's safety and that of other drivers on the road. Research on the impact of mental activities on driving behavior [19] indicate that the method of interaction is the main reason for increased risk, not the task itself. Although web browsing may be a cognitively difficult task, as long as it is accomplished in ways that minimize manual interaction and visual stimulation, it can be done safely.

Safety is of primary importance to our system. The NHTSA found that typing on mobile devices is the highest risk action a driver can take, up to an order of magnitude more dangerous than talking [19]. Therefore, our system uses speech for gathering queries from drivers.

Artificial intelligence (AI) and machine learning (ML) are often used to try and solve problems similar to the ones we are trying to solve. We chose to use the crowd instead for FCB. Using AI or ML would require depending on automatic speech recognition (ASR) to generate transcribed text from the audio, which has an error rate of 17% or greater for simple English queries [20]. Once you have a text query (which may be inaccurate), neither AI or ML can perfectly generate a correct set of actions to take. Results from similar problem spaces show that for easy queries, around 91% of high-level problems can be parsed correctly and broken down into low level actions, but for hard problems this drops to around 65% [20]. Cascading ASR with such a system without additional error correction would decrease 91% accuracy to around 76%. Other approaches using AI or involving the user in a clarification dialog to remove ambiguity achieves success in around 92% of all queries [20], but this greatly increases the latency and increases the amount of interaction needed from the user. We are excited for the potential advantages that using only crowd workers in place of AI can bring, including more quality control and less interaction needed from users. One can imagine other ways of involving AI, such as using the crowd to simply clean up output from an AI system, but we decided to make our system simpler and focus only on crowd work.

This browser is intended to provide results similar to what a driver could get from having a passenger doing proxy browsing on their mobile device. Drivers are already comfortable with having a passenger doing proxy browsing [18], so this approach fits the mental model of how to retrieve information while driving. Additionally, according to the NHTSA, interacting with passengers is one of the lowest risk activities one can do while driving [19]. To accomplish this goal, the crowd worker's interface provides a way for them to have insight into what the driver's needs are. Similarly, the driver's interface allows the user to get a glance at what the crowd worker is working on, just like a driver would be able to get an idea on what page the passenger is on. At points where data is returned or direction is needed, the system uses text-to-speech to read the information aloud, imitating a passenger reading aloud the result he found during his search.

Using human workers also means quality control is needed before returning results. Crowd workers are often paid little amounts of money and do not know the end user, and so they often put in as little work as possible to earn their reward. We have seen in ESP [23] and Adrenaline [2] that this can be solved by using multiple workers in competition or collaboration. This allows us to narrow down large sets of data intelligently using crowd workers, even though there isn't always a single correct answer. We also provide workers with tools specific to the type of action the driver wants to accomplish, which allows them to do better, and quicker, work.

1.2 Scope

We limit the scope of our project to types of browsing that can be accomplished on an anonymous computer. One potential set of solutions would involve personal communications such as using Google Chat or Facebook. We don't address these because of privacy concerns. If we allowed those needs, then the crowd worker interface would need to anonymize the driver's personal information. However, depending on the information need, this could make the crowd worker's job extremely difficult, if not impossible.

Additionally, we do not focus on needs that require time-sensitive manual inputs or produce time-sensitive visual outputs, such as watching videos or playing a Flash game. This exclusion is for safety reasons, as these very clearly would break the NHTSA guidelines [19] for glance length, greatly increasing risk.

According to a survey of mobile users, even with the previous exclusions, we still cover almost 80% of mobile informational requests [18].

Our scope includes queries which follow a "tree" of browsing interactions. Each query is a node in the tree, and each answer provided by workers is a new branch from that node. Drivers can follow branches by clicking on the answers, or create a separate tree by asking a query unrelated to the current line of information. Additionally, drivers are given the ability to go backwards up the tree and revisit previous results.

1.3 FlightCrew Browser

FlightCrew Browser makes it safe for drivers to investigate evolving information needs by using an efficient dashboard interface to get curated results from crowd workers. This requires two interfaces, one for use by the driver and one for the crowd workers.

The driver uses the tablet interface, attached to the car's dashboard, to input queries using speech. It is well known that in order to increase information density on dashboard interfaces while keeping risk low, systems can use multiple input and output methods. We found that visual information should be glanceable, salient and short, while audio output can be more extensive, since drivers can safely listen while focusing on the road. Once answers are ready, we show a glanceable amount of information on screen, as shown in Figure 1. Additionally, the entire answer is read aloud to the driver. The user can click on results that are interesting and get more detailed information.

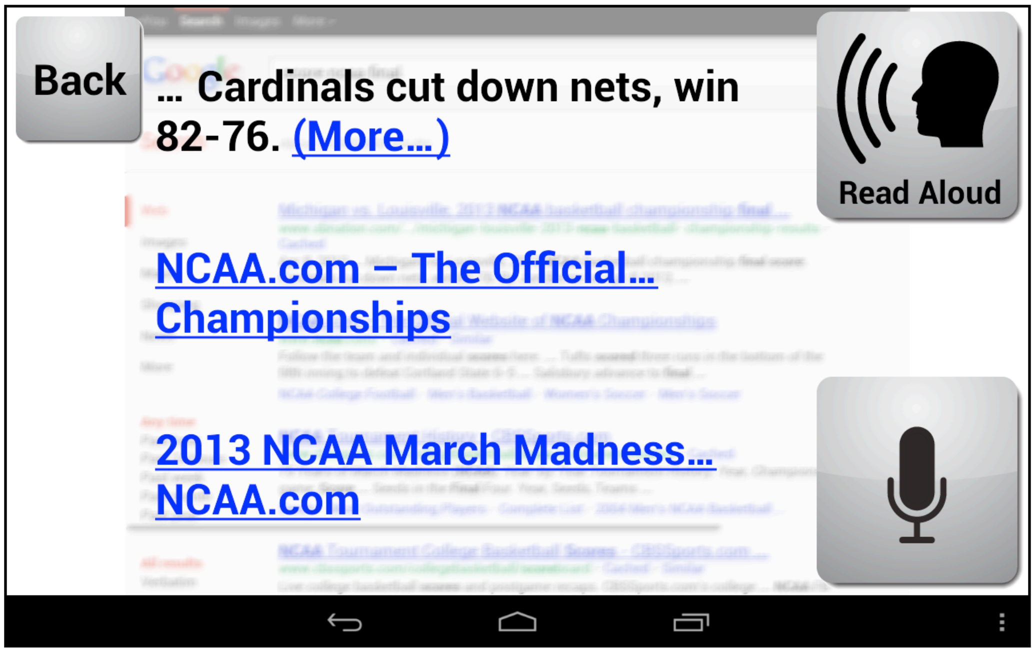
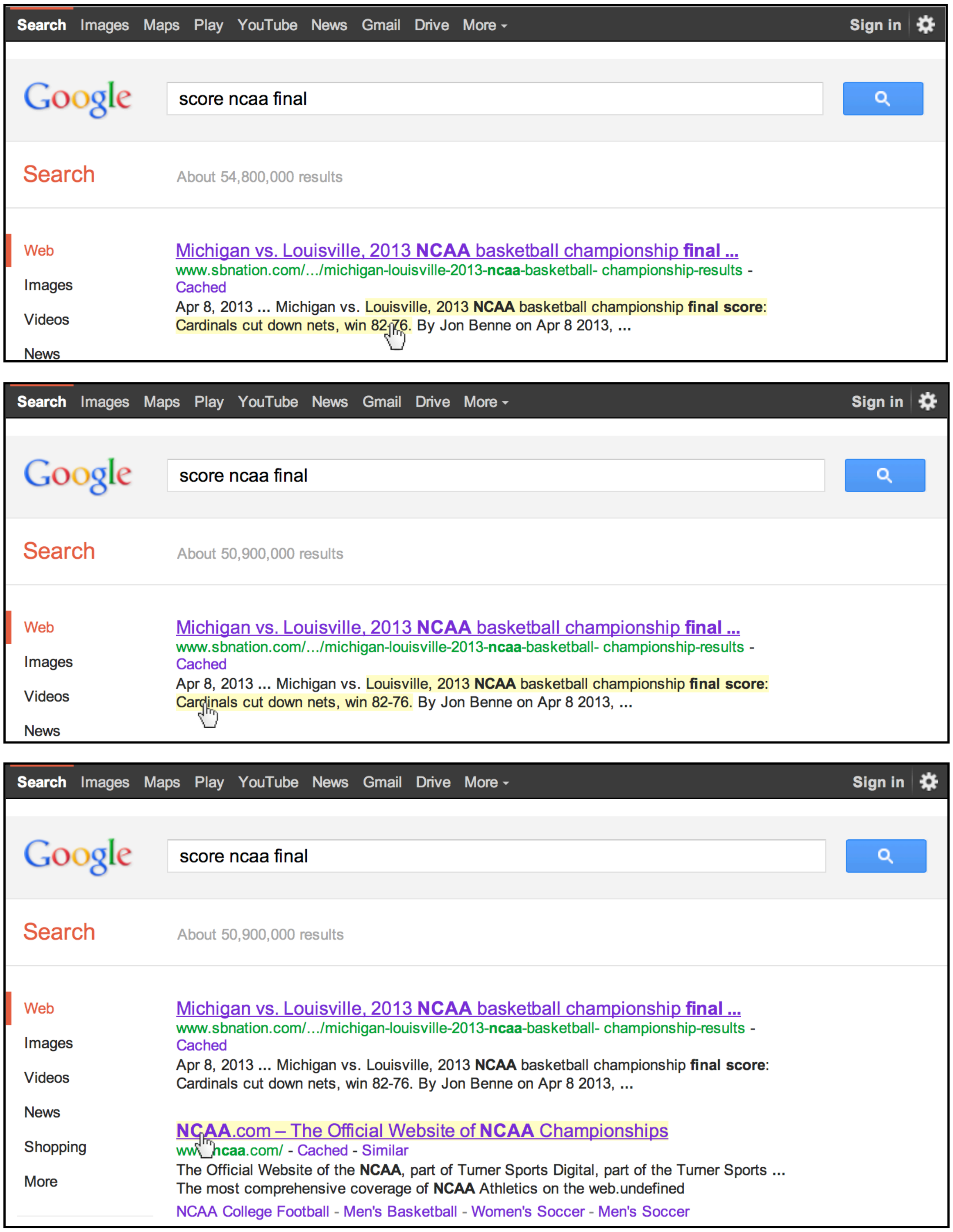

Back
… Cardinals cut down nets, win 82-76. (More…)
Read Aloud
NCAA.com – The Official… Championships
2013 NCAA March Madness… NCAA.com

Figure 1: Answers Displayed for query "What was the score of the NCAA final?"

We found that this combination of inputs and outputs made a large difference in the safety of the interface when compared to the native Chrome web browser. Additionally, our system was significantly more comfortable to use while driving and drivers were equally confident in the correctness of answers shown in FCB, even though they never saw the original page.

Supporting our driving interface is a system on the backend which provides the quality-controlled answers from crowd workers on Amazon's Mechanical Turk [1]. Our crowd interface recruits three workers at a time to answer the queries given by the driver. They are asked to look at a webpage and perform a single action, such as clicking on links, images, and text, or entering new search terms. In order to make affordances for this behavior, we built off of previous results in crowd interfaces that show the ability of crowds to match keywords to an image [23] or progressively narrow a continuous selection [2] using a collaborate game. We extend this concept to allow choosing relevant pieces of data from a webpage, as shown in Figure 2.



score ncaa final

About 54,800,000 results

Michigan vs. Louisville, 2013 **NCAA** basketball championship **final** ...
www.sbnation.com/.../michigan-louisville-2013-**ncaa**-basketball- championship-results - Cached
Apr 8, 2013 ... Michigan vs. Louisville, 2013 **NCAA** basketball championship **final score**: Cardinals cut down nets, win 82-76. By Jon Benne on Apr 8 2013, ...

score ncaa final

About 50,900,000 results

Michigan vs. Louisville, 2013 **NCAA** basketball championship **final** ...
www.sbnation.com/.../michigan-louisville-2013-**ncaa**-basketball- championship-results - Cached
Apr 8, 2013 ... Michigan vs. Louisville, 2013 **NCAA** basketball championship **final score**: Cardinals cut down nets, win 82-76. By Jon Benne on Apr 8 2013, ...

score ncaa final

About 50,900,000 results

Michigan vs. Louisville, 2013 **NCAA** basketball championship **final** ...
www.sbnation.com/.../michigan-louisville-2013-**ncaa**-basketball- championship-results - Cached
Apr 8, 2013 ... Michigan vs. Louisville, 2013 **NCAA** basketball championship **final score**: Cardinals cut down nets, win 82-76. By Jon Benne on Apr 8 2013, ...

NCAA.com – The Official Website of **NCAA** Championships
www.**ncaa**.com/ - Cached - Similar
The Official Website of the **NCAA**, part of Turner Sports Digital, part of the Turner Sports ... The most comprehensive coverage of **NCAA** Athletics on the web.undefined
NCAA College Football - Men's Basketball - Women's Soccer - Men's Soccer

Figure 2: Three workers clicking on text in a webpage for the query "What was the score of the NCAA final?" Agreement is reached for the sentence with "82-76", since two of the workers clicked on it, meaning it is returned to the driver as an answer.

This allows quality controlled selection of HTML elements, such as links, images, and text. We also created an algorithm that, given more than one set of search terms generated by workers based on a human query and potentially imprecise ASR text, picks which of the search terms are most likely to provide appropriate results for the original query. We found that our system provided answers that were more likely to be correct when compared to answers chosen by a single crowd worker, however this came with an increase in latency.

1.4 Contributions

Our work on FlightCrew Browser provides several contributions to the field. We list them briefly here. Each will be described in more detail in the following chapters.

- A novel interface combining text and audio to allow for safe web browsing
- An algorithm to turn a dynamic webpage into a static page while maintaining basic web affordances
- An extension of crowd cooperation games, enabling voting on HTML elements
- An algorithm for choosing between several sets of search terms, using ASR text to provide quality control
- A method for simulating work performed by a synchronous group of crowd workers, using only asynchronous work from those crowd workers
- An evaluation of our mobile browser interface demonstrating higher safety for drivers when compared to built-in interfaces

Chapter 2

Related Work

In this section, we review related work from previous research. We discusses other mobile applications powered by the crowd, constraints imposed on driving interfaces by safety concerns, alternatives to standard web browsing interfaces, ways to control crowd workers to ensure they provide quality results rapidly, and similar work using ML and AI.

2.1 Crowd Powered Mobile Applications

This project has roots in Sinch [15], a mobile interface powered by Mechanical Turk that allows the user to ask a single question and shows multiple answers, leaving the quality control to the end user. This work illustrates that the crowd can be used to give users on mobile devices answers to basic search queries in a timely, glanceable way. FCB attempts to broaden the goals of this project to include more forms of web browsing, not just question-answer sessions, which can be limited in length and scope, as well as provide quality controlled answers. The Sinch thesis presents a system with an interface for the crowd workers and for the end-user. It includes specialized answer pages, which reduce whole pages of information to a couple salient pieces of data that can be easily viewed by a mobile user, and it uses speech as a main interaction method, which is considered critical for in-car use. The FCB system adds specialized worker pages, as well as including quality control instead of simply listing every answer provided by workers.

2.2 Driver Interfaces

It's important to quantify the limitations of drivers when it comes to using non-driving-critical interfaces. The NHTSA detailed important metrics for interfaces used while driving [19]. They found that 17% of crashes involved distracted driving, and that any single distraction for greater than 2 seconds greatly increased the risk. Additionally, tasks done in less than 12 seconds of cumulative distraction time are low-risk, suggesting that interfaces limit their interactions to 6 steps of 2 seconds each. However, the danger to the driver and others greatly depends on the method of interaction. Manual interactions were found to be much riskier than speech interactions, since eye-glances led to visual distraction, which increases the chance of crashing much more than cognitive distraction.

Terken, et. al., evaluated a system using objective and subjective metrics in order to determine how a speech based text entry system would impact distraction and safety [21]. The objective evaluation indicated that there was not a significant increase in mental workload with their speech based system, and drivers had less risk overall. However, the subjective evaluation shows significantly higher levels of stress. They conclude that this higher level of stress was a result of poor automatic speech recognition (ASR), since users could see their incorrect inputs on screen and were able to go back and correct problems. FCB does not show ASR text to users, and the system is not used to compose word-by-word paragraphs for use in an SMS or email message. This allows our system to take advantage of the higher safety levels of a speech based system, while avoiding higher stress levels by using groups of crowd workers to interpret queries and provide quality controlled results.

For touch interactions on in-car driver interfaces, Lasch and Kujala show that distraction while searching through large sets of information varies depending on how the information is arranged [11]. Grouping elements into pages of three elements (compared to groups of five and seven) enables more efficient visual sampling efficiency, but the number of manual interactions is increased since there are more pages of data the user needs to search through. For FCB, we provide up to three quality-controlled results, so we prioritize visual sampling efficiency, which decreases glance length and count. Instead of providing multiple pages for the user to scroll through, drivers can indicate they want more details on one result, or they can clarify their request to get different results. This way, we do not require users to look through large pages of information, and instead rely on the crowd's human intelligence to provide quality results. However, FCB allows the user to have multiple "tabs" open at once, allowing him to explore several information needs at once. The Lasch and Kujala paper also shows that distraction while navigating between screens of data varies depending on the method of interaction. The paper concludes that swiping is less distracting, and thus safer, than page turning buttons, thus FCB allows changing tabs with swiping gestures.

The typeface used in interfaces impacts how efficiently users can read it, which is especially important for driving. Reimer, et. al., showed that optimizing typeface characteristics is one way to reduce the mental demand and level of distraction for an interface used by a driver [16]. They conclude that humanist typefaces (those with more calligraphic shapes) lower visual demand by up to 12.2% compared to grotesque typefaces (those with more precise, abstract shapes), where visual demand is a measurement combining glance time, glance frequency, and number of long glances over 1.5 seconds. FCB uses a humanist font, though because it uses a combination of speech and visual output, we do not expect as large of an improvement.

2.3 Web Browsing

Law and Zhang show that when users have an information need, the goal they want to accomplish can be described in a high-level mission statement, but these do not necessarily translate directly into useful search queries for current search engines unless rewritten intelligently [12]. CrowdPlan solves this by providing high-level queries and contextual information from the user to a crowd worker, at which point the crowd worker transforms that into a query that is usable by a search engine. For FCB, the driver is providing audio queries and the contextual information includes the fact that the user is driving, where the user is located, and the current time. Crowd workers interpret the driver's needs and perform actions that solve the information need, such as transcribing the query and synthesizing it into a new search or selecting HTML elements on a webpage that answer the question.

Collaborative browsing connects two users to allow them to share data and a browsing session. This often requires one of the users to direct or observe the other, non-collocated, user doing the majority of the browsing. PlayByPlay [25] is a system that provides easy affordances through a chat interface for many of the common operations that a user, mobile or stationary, requires during collaborative browsing. The two browsing sessions are not necessarily parallel, but both users' progress is tracked in a chat UI. Results from PlayByPlay show that manual interaction, such as clicking with a mouse or using a touch interface, is the most distracting interaction scenario for mobile users, and that visibility of previous actions taken by both users is critical to ease-of-use. This system is a general purpose collaborative web browsing solution, where FCB intends to provide a similar solution under the constraint that one user is driving a car, and so the ratio of browsing to providing instructions is more one-sided. As opposed to using a single remote user, FCB uses human crowds to increase quality control and decrease latency. FCB provides a way to go backwards in the session if needed, but doesn't show the full history to the driver.

Desktop web browsing involves looking at large amounts of data, such as web pages or forms, and users often want this data transformed in ways such that the page can be easily viewed on a mobile interface. Research has found several solutions to this, and each can be classified as manual solutions with additional tools, or automatic solutions with heuristics or machine learning. The manual solutions, like PageTailor [5], are systems where users are given a toolbar, or a design application, which allows them to transform pages by hand into more succinct versions. This same system could be used by crowd workers and the results sent to end-users. Compared to automatic solutions, manual methods generally achieve better results, since a human is doing the work, but the increase in latency is not something we can depend on for a system intended for drivers, especially if the workers need to be trained to use special tools, which would increase latency even more. WebAlchemist [24] and Bricolage [10] are two systems that use algorithms to convert a web page to a version that is usable on a mobile device, and are likely to produce reasonable results with low enough latency. WebAlchemist uses heuristics and Bricolage uses machine learning. FCB is not able to depend entirely on algorithmic approaches to simply display the entire page to the driver since the informational needs will vary over browsing sessions, and the returned results should be a minimal answer to the query, removing extraneous page elements.

Helping vision-impaired users experience the web often involves webpage transformations similar to those already discussed. HearSay3 [6] is a browser designed to improve the web browsing experience for vision impaired users. It provides a segmentation technique to "layer" web pages to allow the user to move between the segments quickly, without seeing the page itself. It also provides a system of voice-commands, which is used to automate some of the browsing experience. Many companies also provide general solutions to this problem for entire computer systems. One successful system is VoiceOver [22], created by Apple. VoiceOver is built into every iOS and Mac OSX product, but they are complicated to use since they try and solve many problems at once. Additionally, users are often committed to using the system very frequently so they can take the time and effort to become extremely efficient with all the commands and tools, which is not be the case for our system. Since our users have the ability to use both touch and speech input, we focused on using input methods that make sense for the actions the user intends to take. For example, inputting queries is done using speech, but indicating interesting pieces of data that should be clarified or expanded is done using touch input directly on the piece of data itself.

2.4 Crowd Control

In order to achieve low latency results when using asynchronous crowds, workers need to be available immediately after the user makes a query. This can be solved using the retainer model introduced in Adrenaline [2], which recruits workers before the end-user has made any requests, and gives them a bonus for being punctual at returning to the task when the request arrives. This ensures that a set of workers will be ready when the end-user wants them, as well as get them working quickly. Our system takes advantage of this, and additionally shows instructions and examples while workers are waiting, which decreases latency once a task is available.

Once workers are recruited, they need to produce results that match the end-user's needs. There is usually a trade-off here between latency and quality; by decreasing latency, quality is often sacrificed. However, if more than one worker is used, they can collaborate and verify each other. Smart Splitting [14] is a technique used when performing searches or inspecting full pages where results are distributed among several workers to get a decision with a smaller latency. However, these results would need to be verified by more crowd workers, since each answer isn't chosen with the full context of the page and thus the best result may not be accurate. We have found that verification can be done in a competitive way while the results themselves are being chosen; this will be discussed in greater detail later. ESP [23] shows that two workers coordinating to match keywords for an image is a great way to get quality controlled results quickly. FCB puts three crowd workers to work on a single query by making them vote on important pieces of data on the page. This narrows down whole pages to at most three pieces of data most relevant to the end-user’s query. The task is only concluded when workers agree on a best answer, but more than one answer is actually returned to the user. This encourages each worker to choose more than one piece of data, sometimes even more than three. This improves the variety of responses, as well as decreasing latency, as workers know they are coordinating with another human being.

Browsing sessions are often more involved than a single piece of input and output data. VizWiz [4], a system allowing blind users to take pictures of a scene and ask crowd workers for information about it, found that both mobile users and the crowd workers desired more interaction beyond a single question and answer. Their study suggested that interactive communication is important to increasing confidence in answers and would greatly enhance the experience for both types of users. Since FCB is using asynchronous crowds, we are not able to ensure that the browsing session involves the same workers every time, but the system provides a limited history of interactions, enabling contextual information about the current query at hand. Additionally, the system provides more than just speech output for the end-user, allowing for more contextual information.

2.5 Artificial Intelligence and Machine Learning

Many systems sharing goals with FCB have used AI and ML to solve latency and efficiency issues, as algorithms usually provide answers quicker than human-driven systems. However, there are many issues with these approaches. Recent results from Google on ASR [7] show that word error rate for is still at 17% or higher for transcribing news broadcasts, and goes up to 50% or higher when transcribing YouTube videos.

There are many commercial systems that use such algorithms to provide question and answer sessions or information retrieval. Siri [17] and Google Now [9] both take voice queries and use ASR to turn them into textual queries, before running them through backend systems to retrieve data and information. In addition to ASR issues, they are not adaptive to queries that evolve as the user progresses. This means once the system has an answer to a question, it is not usually possible for a user to ask tangential questions and have them pick up exactly what was meant. Siri is very good at dealing with information that built-in Apple apps can access, and it can poll several web services for data online. However, it fails on follow-up questions. For example, asking “Who is Jay-Z?” pulls up a page on him and his life, which is accurate. But asking “Who is his wife?” immediately afterwards fails, and Siri points you to a web browser. FCB succeeds in questions like this by using human intelligence and showing the workers some of the previous context.

Both Siri and Google Now resort to showing a mobile browser interface on questions that they don’t immediately know the answer to, introducing the same safety issues that exist with those interfaces. Even on queries that are imprecise or hard for crowd workers to answer, FCB always puts the safety of the driver first and provides ways for them to correct or guide the crowd as necessary, instead of showing them an unsafe, built-in mobile browser.

Chapter 3

Design

In this chapter we present the design and the principles that motivated both the driver and the crowd interfaces.

We chose proxy browsing with a passenger in the car as the model for the types of interactions a driver expects out of FCB. As discussed in Section 1.1, this is supported by studies that show that 76% of mobile information needs are solved by asking another human, using the web, or directly contacting the source of the information [18]. The majority of the other needs were solved by printing information beforehand or going to the location directly, both of which do not fit the scope of types of information we are trying to cover with FCB. Using online maps is the last category mentioned in the paper, which accounted for 10% of the described needs. Some, but not all, of these queries are accomplishable within the scope of our project, so we are not using this need to motivate our design.

3.1 Driver Design

Our Android tablet application is focused on allowing the user to make a query and direct the crowd workers on the type of information they want. We designed a simple interface with only two types of screens: the initial screen which only has one interface element, and the results page, shown in Figures 3 and 4, respectively. The only element on the initial screen is a microphone to make a query, and is located in the same place as the microphone on the results page.

The safety of the driver was the most important factor we considered in our design. According to the NHTSA, limiting the amount of information shown to the driver at one time is of critical importance [19]. The results screen only shows the top three results (selected by the crowd workers) from any single page, and it only shows portions of the text for each result, as shown in Figure 4. In order to still allow users to get the information they need, FCB reads aloud the full text of the result as it arrives, since listening to speech doesn't distract the driver. If a driver misses something, or just wants to hear the results again, there is a button that causes the application to read all of the answers aloud. Additionally, we provide a "blurred out" view of a website, shown in Figure 5, in the background of the results screen and overlay the rest of the interface over that. This gives the driver context so they know where the data came from, while still allowing them to focus on the actual results.

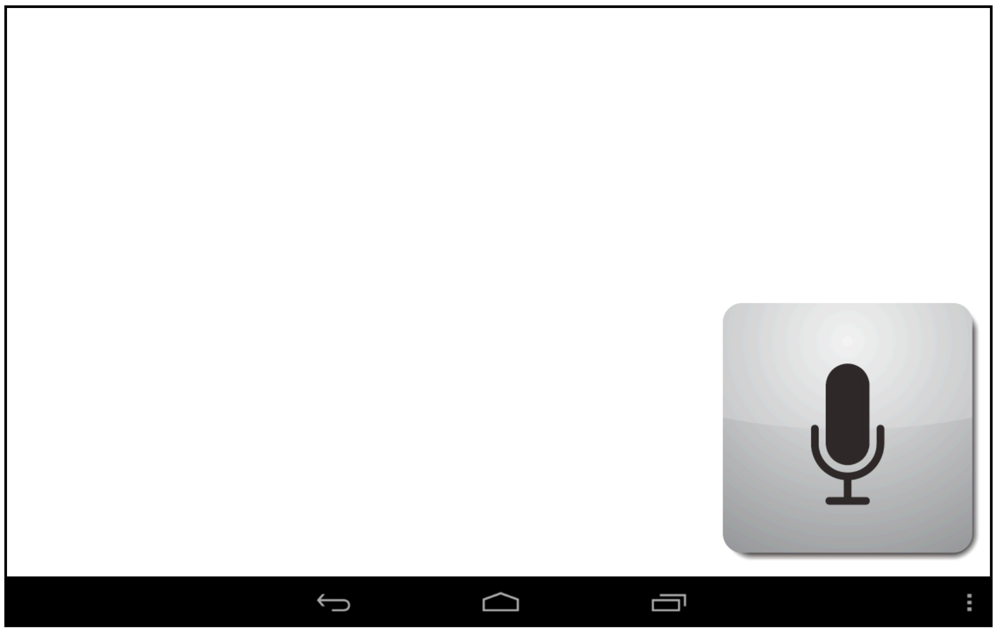


Figure 3: Initial screen drivers see

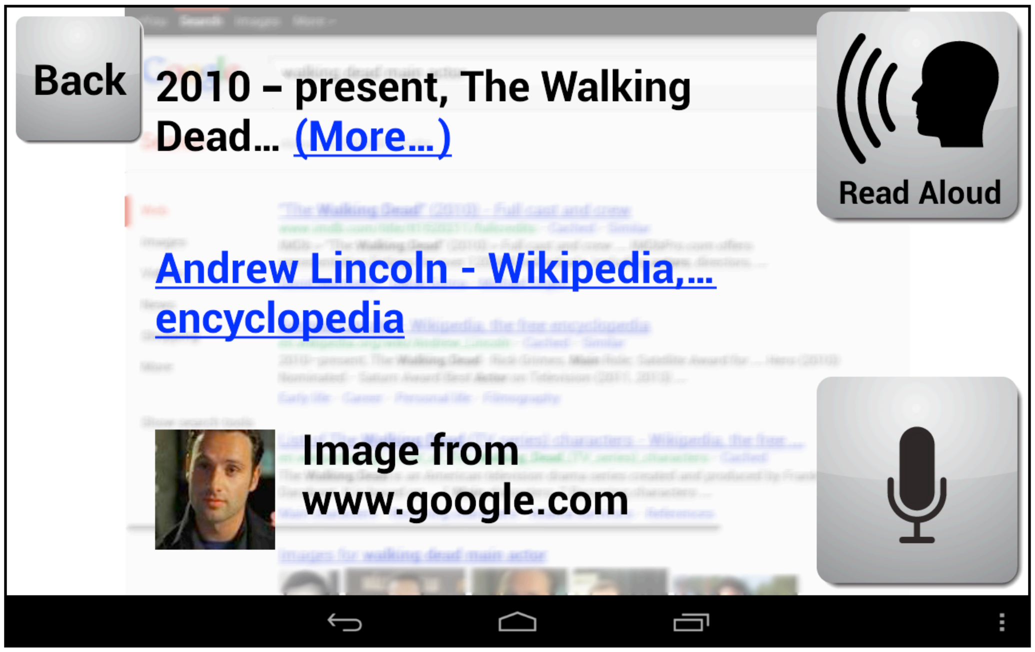

Back
2010 – present, The Walking Dead… (More…)
Read Aloud
Andrew Lincoln - Wikipedia,… encyclopedia
Image from www.google.com

Figure 4: Answers displayed for query "What else has Rick from Walking Dead been in?"

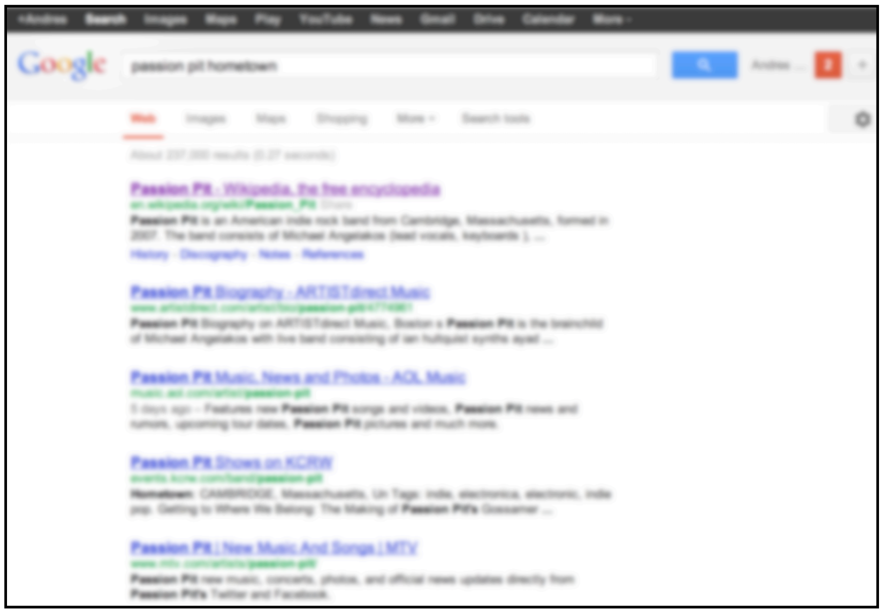


Figure 5: Blurred out version of a website as background for the results screen for query "Where is Passion Pit from?"

We kept the scope for FCB small and focused on the actions users most often take while web browsing. These actions are: making queries, clicking on a link, looking at an image, and reading important text. Users are accustomed to touching or clicking on elements they are interested in, so we designed our interface to allow users to directly touch elements they see on screen to see more information, as is shown in Figure 4. For images, this shows the picture fullscreen. For links, the system creates a followup query, re-asking crowd workers the current query on the page linked. In the case of clicking on the second answer in Figure 4, this would ask "*What else has Rick from Walking Dead been in?*" to workers while showing Andrew Lincoln's Wikipedia page. For other elements (text, headings, etc), clicking on them shows more context around that element. For instance, in Figure 4, clicking on the first result will show the sentence before and after the HTML element where the crowd worker found that answer.

Based on studies from the NHTSA, if information disappears without direct action from the driver, risk is greatly increased. We therefore never remove information unless the user presses a button or swipes to a new screen.

Pressing the microphone button starts recording audio, allowing drivers to submit new queries. It can handle both context-independent and context-dependent questions, since crowd workers are able to see the web page that the last results for this query came from. This makes it easier for the driver as they don't have to use multiple buttons, one for followup questions and one for new queries.

There is a back button, which works similar to how a back button in a web browser works; it takes you back one step in the set of results found. For instance, if you were looking at Andrew Lincoln's Wikipedia page and pressed the back button, you would see the three Google results as shown in Figure 4.

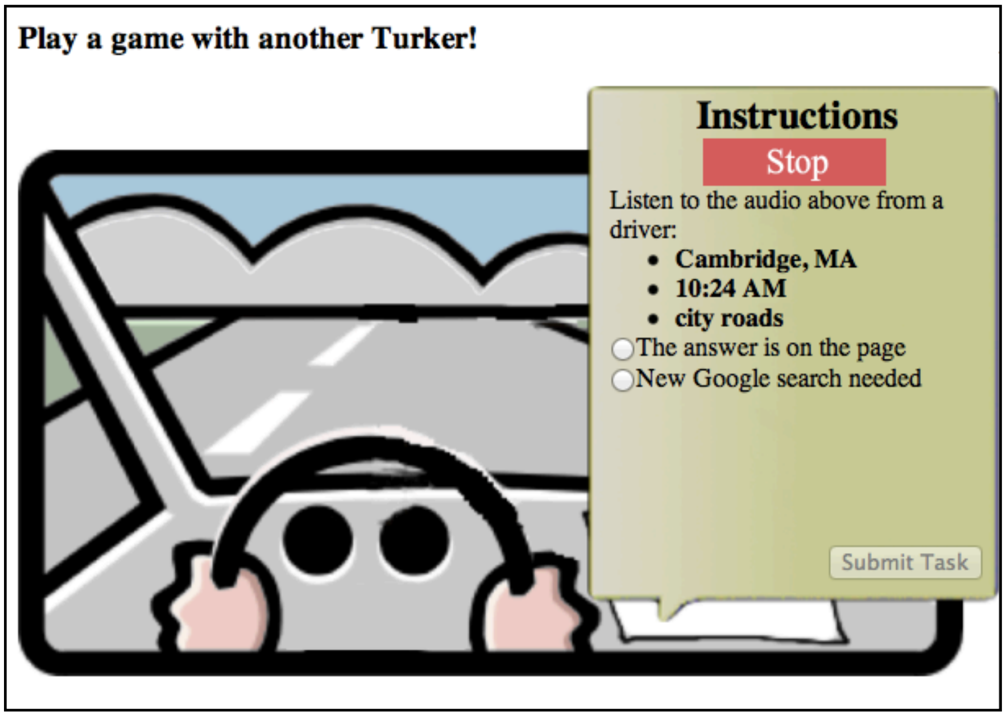
Another important consideration for a driving interface is efficiency, which can be measured in several ways. For the driving interface, there are two conflicting measurements of efficiency: the amount of time a user spends glancing at the screen, and the amount of time before the user gets the information they need from the screen. These are conflicting for our use case because of the limited attention we can demand from the driver. Decreasing the glance length can be accomplished by removing information from the screen and reading it aloud instead. As noted previously, we limit the amount of text shown for each result and we automatically read aloud each result as it comes in, which means users do not necessarily need to read anything on screen to get new information, only to orient themselves as to which answer is which. On the other hand, this increases the latency for retrieving the information they need, as listening is inherently slower than reading, especially if there is additional contextual information that could be skimmed visually, but not aurally. Because of safety concerns, we focused on decreasing glance length, not information retrieval times.

3.2 Crowd Interface

The design of our crowd interface is intended to return accurate results as quickly as possible. As discussed in Section 1.2, we focused on information needs that can be found by an anonymous worker, i.e. from Mechanical Turk, but our system could be extended to support other crowds and information needs as well. Our crowd interface has two aspects: the instructions and the task interface itself. Workers are expected to read the instructions and then use the task interface to give the system an answer.

Modeling our interaction pattern off of proxy browsing means the instructions must establishing common ground between the crowd worker and the driver. This is accomplished in two ways. In order to ensure the Turkers know the user they are providing answers for is a driver, we show the instructions (including the query itself) inside a speech bubble coming from the perspective of a person inside of a car, as shown in Figure 6. Additionally, the driver's specific context is shown bold and bulleted. This includes the driver's location, current driving pattern, and local time.

Another challenge is supporting evolving information needs, i.e. those without just a single query and single answer. We include the page which the driver was last looking at when he made the new query, as in Figure 7, which helps crowd workers interpret contextual queries successfully. For example, asking "who is the lead actor" without context is not answerable. However, if the driver had just asked "when is the next walking dead," and is looking at a webpage with information on that TV series, the query becomes much clearer.

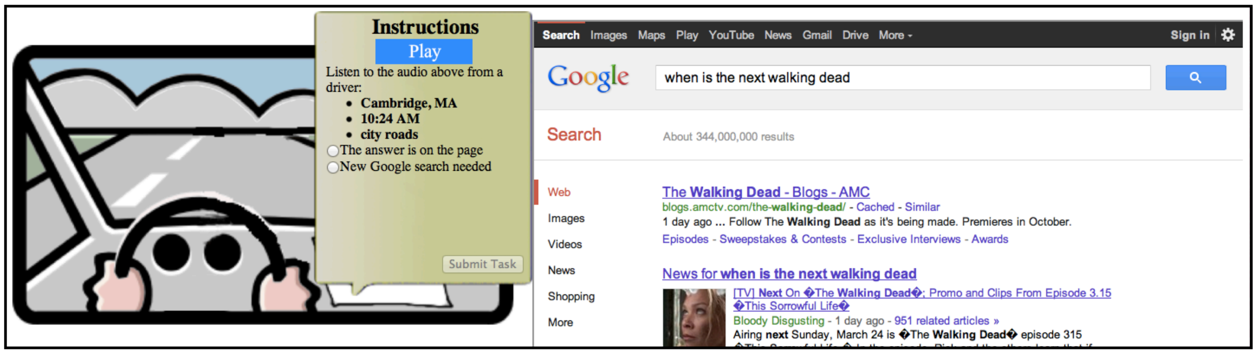


Listen to the audio above from a driver:

- **Cambridge, MA**
- **10:24 AM**
- **city roads**

◯ The answer is on the page
◯ New Google search needed

Figure 6: Instructions for crowd workers



Listen to the audio above from a driver:

- **Cambridge, MA**
- **10:24 AM**
- **city roads**

◯ The answer is on the page
◯ New Google search needed

when is the next walking dead

About 344,000,000 results

The **Walking Dead** - Blogs - AMC
blogs.amctv.com/the-**walking-dead**/ - Cached - Similar
1 day ago ... Follow The **Walking Dead** as it's being made. Premieres in October.
Episodes - Sweepstakes & Contests - Exclusive Interviews - Awards

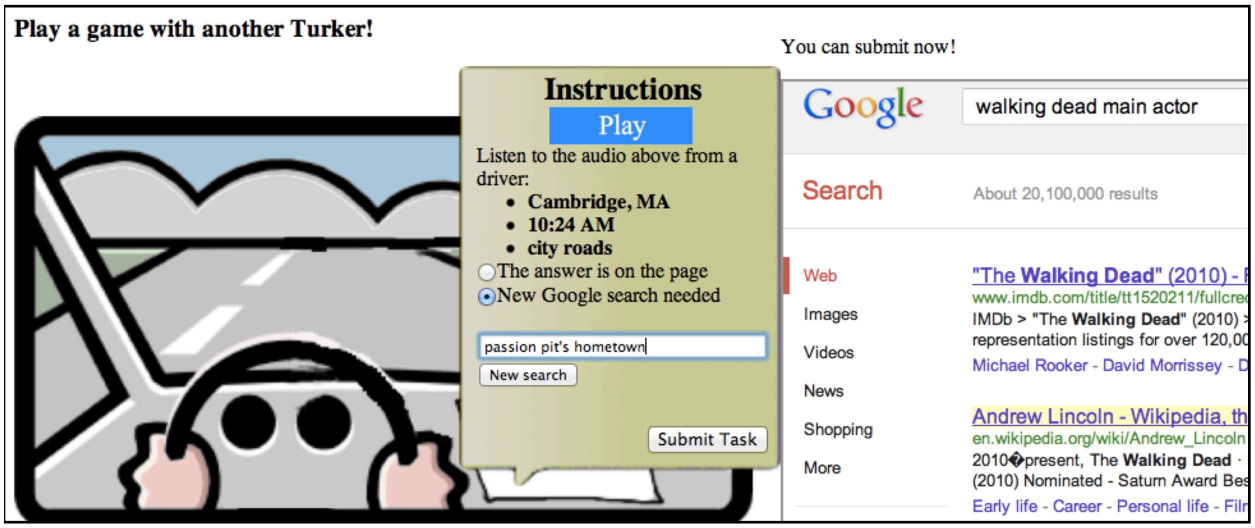
News for **when is the next walking dead**

[TV] **Next** On �The **Walking Dead**�: Promo and Clips From Episode 3.15 �This Sorrowful Life�
Bloody Disgusting - 1 day ago - 951 related articles »
Airing **next** Sunday, March 24 is �The **Walking Dead**� episode 315

Figure 7: Instructions for crowd workers and current webpage

Our task interface asks the workers to pick between making a new search or clicking on answers on the page, as is shown in Figure 7. We have seen that approximately one of three Turkers do not perform well on tasks [3], and so we designed our system to recruit three workers for every task in order to help with quality control. This means that for each task, we can expect only two workers to make the correct choice between a new search or in finding the answer on the page.

If workers decide that the query should lead to a new search, they are asked to input search terms as if they were doing a Google search, as shown in Figure 8. Workers are accustomed to inputting terms into search boxes and as discussed in Section 1.1, humans are good at turning high level information needs into queries that search engines understand well. Once we have more than one set of search terms, we run an algorithm (discussed more in Section 4.4) to decide which set of terms are better and make a new Google search with them. We then start the process over with new workers. They are shown the new page with the previous query, so the answer should be on the new Google results page, and we won't need new search terms.


Play a game with another Turker!

You can submit now!

Instructions

Listen to the audio above from a driver:

- **Cambridge, MA**
- **10:24 AM**
- **city roads**

The answer is on the page [✓ New Google search needed]

passion pit's hometown

walking dead main actor

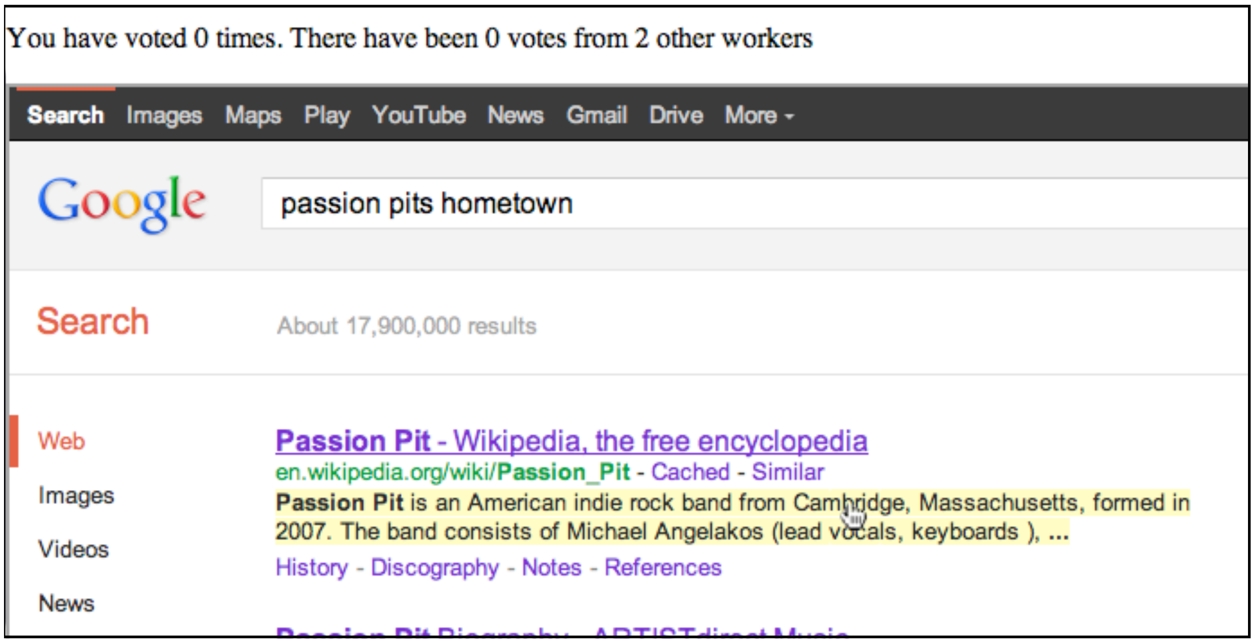
About 20,100,000 results

"The Walking Dead" (2010) - I
www.imdb.com/title/tt1520211/fullcre
IMDb > "The **Walking Dead**" (2010) :
representation listings for over 120,00
Michael Rooker - David Morrissey - D

Andrew Lincoln - Wikipedia, th
en.wikipedia.org/wiki/Andrew_Lincoln
2010�present, The **Walking Dead** ·
(2010) Nominated - Saturn Award Bes
Early life - Career - Personal life - Filr

Figure 8: a worker inputting search terms for query "Where is passion pit from?"

If the workers decide the answer is on the current page, then they are asked to click wherever they see the answer. Most people are accustomed to web browser affordances, so we designed this interaction method off of normal web browsing actions. In the instructions, workers are encouraged to click on the answer if they see it on the webpage adjacent to the instructions. There are other approaches we could have taken, such as providing tools that let workers copy and paste text to provide answers. This was done in Sinch [15], but it requires teaching workers to use the tools. Additionally, doing quality control on arbitrary chunks of text is potentially as difficult as finding the text in the first place. This is because it must be done by humans, as arbitrary text may be equivalent to humans but very different to text similarity algorithms. Instead, we preprocess the webpage and make elements clickable by the user, as in Figure 9. This leverages the existing affordances the web offers to click on links and images, and provides highlighting on text to show what the worker is clicking on.


You have voted 0 times. There have been 0 votes from 2 other workers

Passion Pit - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Passion_Pit - Cached - Similar
Passion Pit is an American indie rock band from Cambridge, Massachusetts, formed in 2007. The band consists of Michael Angelakos (lead vocals, keyboards), ...
History - Discography - Notes - References

Figure 9: Clicking on text in a webpage for the query "Where is passion pit from?"

As workers click on elements, we wait until two workers click on the same element, at which point we send this answer to the driver's interface. This provides quality control similar to the ESP game [23]. As was seen in that paper, this cooperative element also encourages workers to work quickly and accurately. To further encourage that, we have a timer incrementing as time passes. The color changes as time goes on, giving feedback to workers that they should be working quickly, as shown in Figure 10.

Finally, in order to assure crowd workers that their clicks are being recorded, we highlight each element they click on and display it above the webpage in a header, as shown in Figure 11. As other workers make their own votes, we indicate that in the header as well, further indicating that other people are working with them.

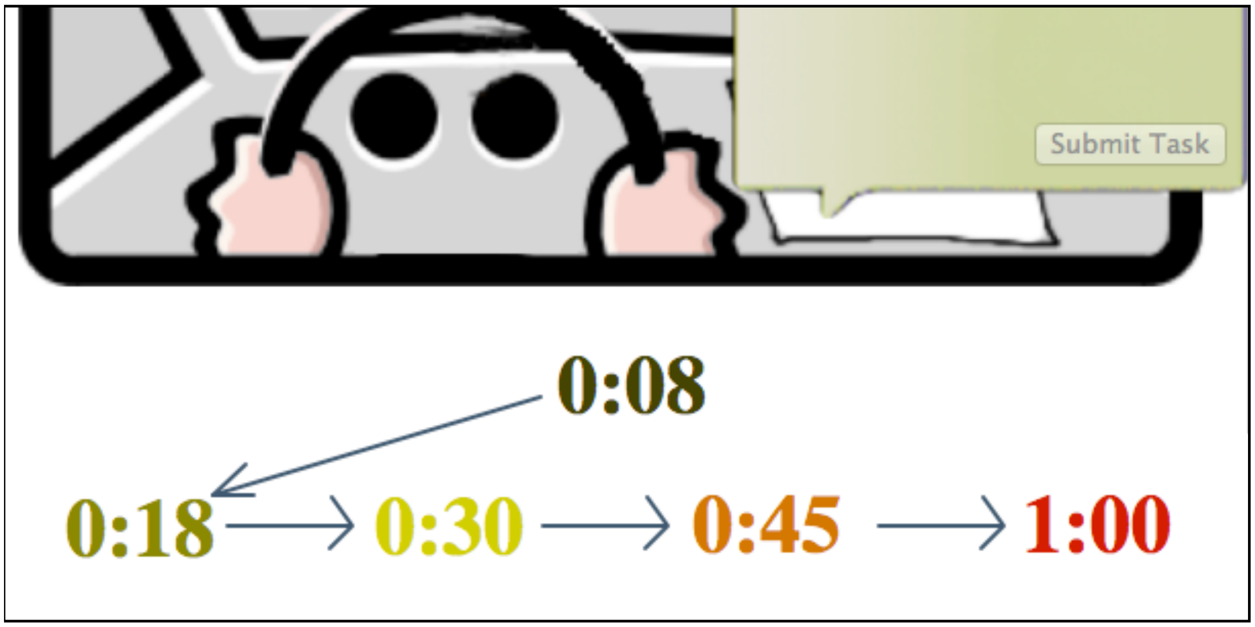
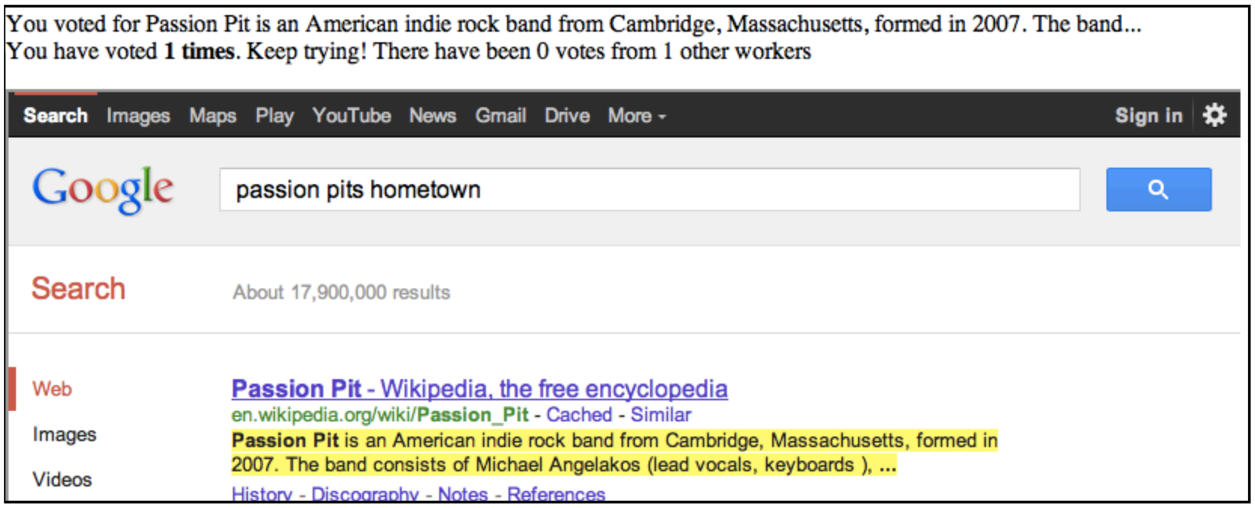


Figure 10: The timer increments, and changes color as it does so, suggesting quick work is preferred



You voted for Passion Pit is an American indie rock band from Cambridge, Massachusetts, formed in 2007. The band...
You have voted **1 times**. Keep trying! There have been 0 votes from 1 other workers

Passion Pit - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/**Passion**_**Pit** - Cached - Similar
Passion Pit is an American indie rock band from Cambridge, Massachusetts, formed in 2007. The band consists of Michael Angelakos (lead vocals, keyboards), ...
History - Discography - Notes - References

Figure 11: Vote is displayed in header, as is a count of other votes

Chapter 4

Implementation

4.1 Overall System

The FCB system has three parts: the Android app, the retainer system, and the server controller. The retainer system controls the workers, the Android app displays data to, and records data from, the driver, and the server controller manages the interaction between the other two. In Figure 12, the flow of information in the system is diagrammed from a high level.

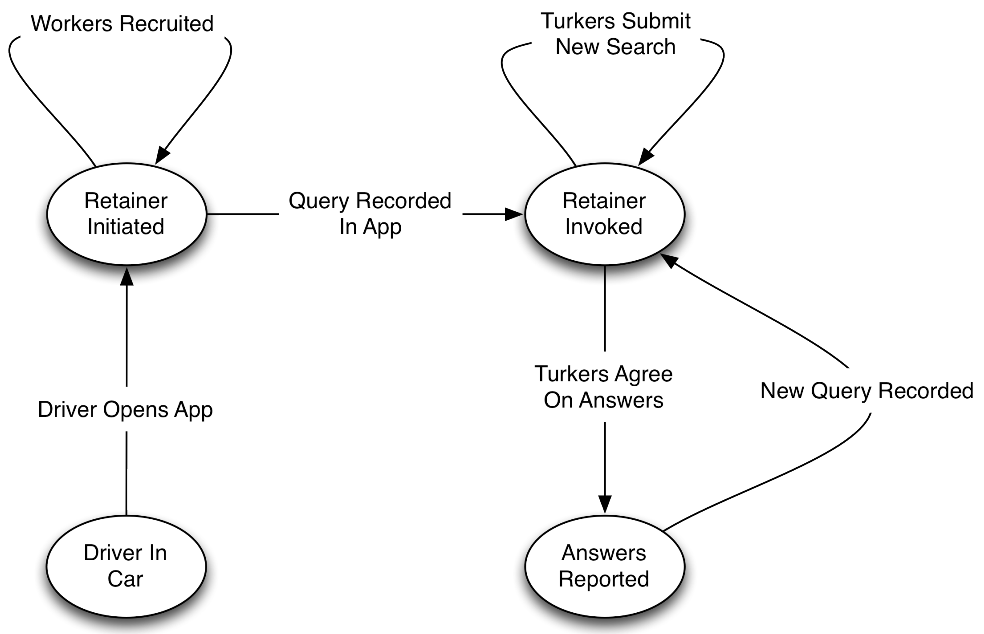


Figure 12: Overall System Flow

When the app is opened for the first time, the retainer system starts recruiting workers, and keeps a pool of them around, ready to start working once a query is submitted. Once there, workers are shown instructions and told to work on other tasks until ours is ready. The Android app records a question posed by the driver, and sends notifications up to the server controller, which notifies the retainer system that work is now available. The workers are taken to the task page, shown in Figure 7, and are told to listen to the query and make an action: click on answers on the webpage or input new search terms (shown in Figures 9 and 8 respectively). If the workers decide a new search is required, the driver is shown a new, still blurred out, background webpage (as shown in Figure 5) and a new task is posted through the retainer system. This new task has the same audio query, but shows the new search results page, so workers can progress towards finding the answer to the information need. If the workers click on elements on the webpage, the quality-controlled answers are reported to the driver. The rest of this chapter explains each piece of the system in more detail.

4.2 Server Controller

The server controller keeps records of the current state of the system and manages all interactions between the Android app and the retainer system. The current state of the system is stored on a MySQL database. There is a set of PHP pages that provide an API for accessing and modifying the contents of the database and controlling the retainer system. The database is shared with other crowd-powered mobile apps, and has shared tables to hold information about queries, device information, and audio files. However, each app has its own tables for data called "events" and "answers", and this is where customization is done to allow for extended functionality within our crowd-powered mobile apps.

Events are recorded in order to mark an occurrence that affects the state of the system. For FCB, we have several types of events that are logged and used to inform both the retainer system and the Android app of updated information. They are listed below with a description of their use and what they effect:

- newQueryAudioUploaded
    - Sent by the Android app; listened for by the retainer system.
    - This indicates that a new query has been uploaded, and that it was recorded from the initial screen (as shown in Figure 3). This causes the retainer system to place workers on the new task.

- unconfirmedAudioUploaded
    - Sent by the Android app; listened for by the retainer system.
    - This indicates that a new query has been uploaded, and that it was recorded from a previous answer page (as shown in Figure 4). This causes the retainer system to place workers on the new task.
- followLink
    - Sent by the Android app; listened for by the retainer system.
    - This indicates that the driver clicked on a link. This causes the retainer system to create a new worker task, with the previous audio and the webpage associated with the link.
- audioConfirmedNewQueryEvent
    - Sent by the retainer system; listened for by the Android app.
    - This indicates that the workers agreed to submit new search terms. This causes the Android app to create a new "tab" in order to keep this query separate from the one where the query was recorded.
- audioConfirmedFollowupEvent
    - Sent by the retainer system.
    - This indicates that the workers found the answer to the query on the webpage, and indicated as such by clicking on HTML elements (as shown in Figure 9).
- newURLToShow
    - Sent by both systems; listened for by both systems.
    - This indicates that the workers are starting a task on a new webpage, either because the driver clicked on a new link or the workers decided to submit new search terms. This causes the Android app to change the background to reflect the new page (as shown in Figure 5).

Answers are responses from the workers showing data that is believed to answer the driver's query. These are sent from the retainer system after workers agree with each other. They are all listened for by the Android app and shown to the user afterwards. There are several types of answers, and each is described in Section 3.1 and shown in Figure 4.

4.3 Android App

The Android interface is based on a previous implementation of Sinch [15], but adapted to web browsing. The Sinch application also evolved into the FlightCrew platform, which includes shared files for all of our crowd-powered mobile apps. FCB includes the FlightCrew platform Android library in order to interface with the CRUD API that allows for sending and receiving data from the server controller. The components of the app are diagrammed in Figure 13 at a high level.

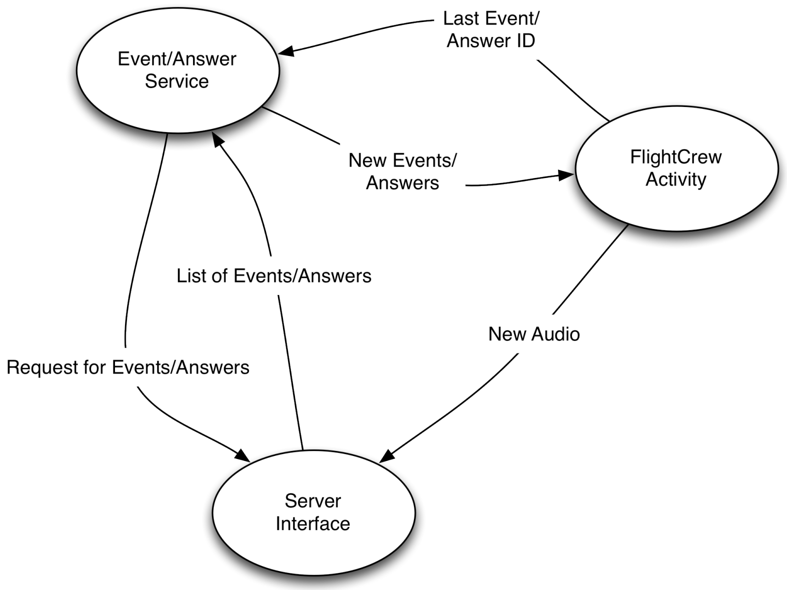


Figure 13: The three major components of the FlightCrew platform used by the FlightCrew Browser Android app

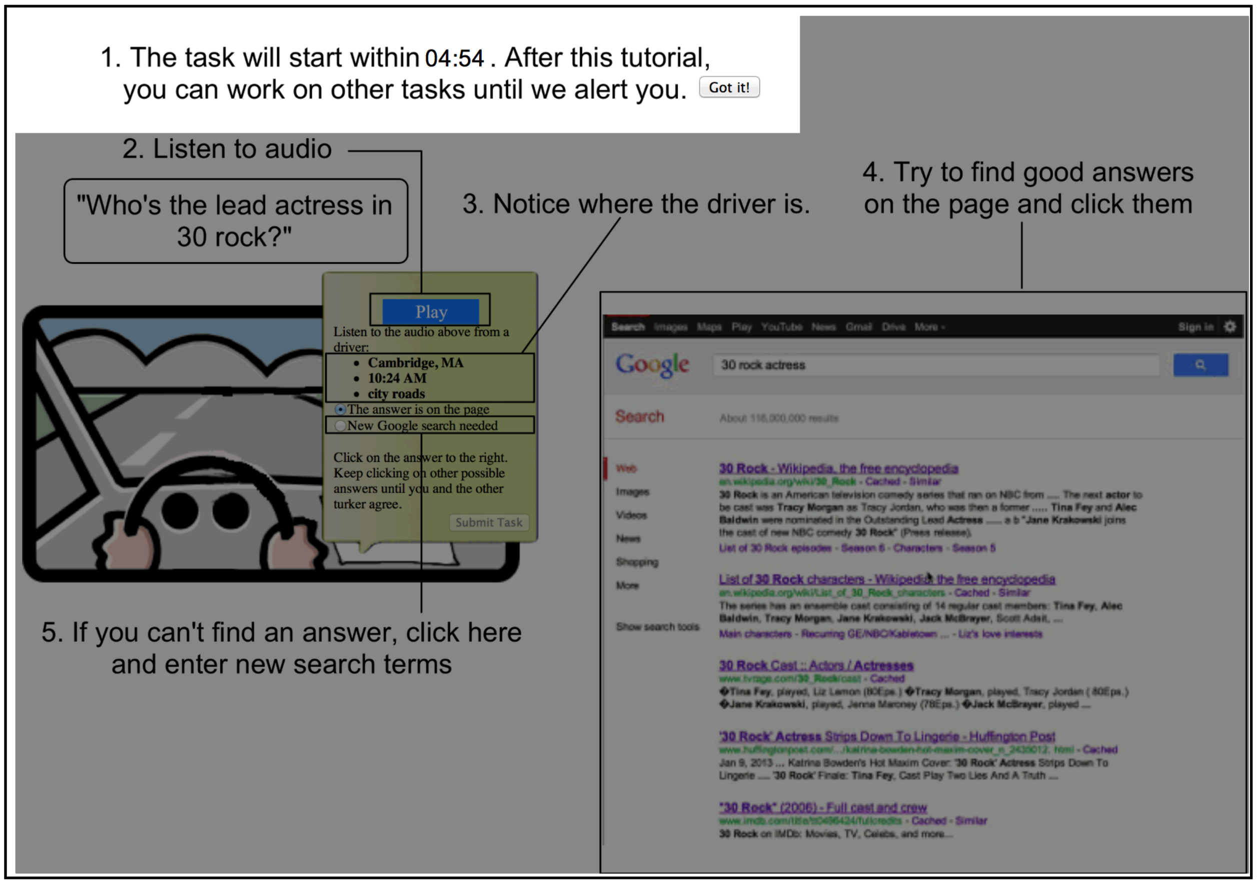
The FlightCrew Activity is the main Android object controlling the app. It is active when the user has the app open on the tablet, but it may be destroyed when the user does not have the app open. It keeps a record of it's current state when closing so when it is reopened, the user is back to where they were last time the app was opened. When the app is opened for the first time, it starts up the event and answer service. This service, unlike the activity, remains alive even if the user closes the app. Both objects have access to the server interface, which is a synchronous wrapper around HTTP requests to the server controller, which was described in Section 4.2.

The service frequently queries the server interface to retrieve any new events and answers since the last query. It stores these and waits until the activity requests them, at which point it sends all of the events and answers it has stored, then removes them. When the activity is alive (the app is opened), it frequently queries the service asking it for any new events or answers by sending the last seen event ID and answer ID.

When a user records a new query, an event is created through the server interface, which causes the retainer system to prompt workers to begin working, as described in Section 4.2.

4.4 Retainer System

The retainer system manages the crowd workers. Once the Android app is opened, the retainer system begins to recruit workers. It attempts to maintain 3 workers waiting for work. While the workers are waiting, they are shown instructions, walking them through the steps needed to do the task. The first step is shown in Figure 14.



1. The task will start within 04:54 . After this tutorial, you can work on other tasks until we alert you.

2. Listen to audio

"Who's the lead actress in 30 rock?"

3. Notice where the driver is.

4. Try to find good answers on the page and click them

5. If you can't find an answer, click here and enter new search terms

Listen to the audio above from a driver:

- **Cambridge, MA**
- **10:24 AM**
- **city roads**

[✓ The answer is on the page]
New Google search needed

Click on the answer to the right. Keep clicking on other possible answers until you and the other turker agree.

30 rock actress

About 116,000,000 results

30 Rock - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/30_Rock - Cached - Similar
30 Rock is an American television comedy series that ran on NBC from ..... The next **actor** to be cast was **Tracy Morgan** as Tracy Jordan, who was then a former ..... **Tina Fey** and **Alec Baldwin** were nominated in the Outstanding Lead **Actress** ..... a b "**Jane Krakowski** joins the cast of new NBC comedy **30 Rock**" (Press release).
List of 30 Rock episodes - Season 6 - Characters - Season 5

List of 30 Rock characters - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/List_of_30_Rock_characters - Cached - Similar
The series has an ensemble cast consisting of 14 regular cast members: **Tina Fey, Alec Baldwin, Tracy Morgan, Jane Krakowski, Jack McBrayer,** Scott Adsit, ...
Main characters - Recurring GE/NBC/Kabletown ... - Liz's love interests

30 Rock Cast :: Actors / Actresses
www.tvrage.com/30_Rock/cast - Cached
�**Tina Fey**, played, Liz Lemon (80Eps.) �**Tracy Morgan**, played, Tracy Jordan (80Eps.) �**Jane Krakowski**, played, Jenna Maroney (78Eps.) �**Jack McBrayer**, played ...

'30 Rock' Actress Strips Down To Lingerie - Huffington Post
www.huffingtonpost.com/.../katrina-bowden-hot-maxim-cover_n_2430012. html - Cached
Jan 9, 2013 ... Katrina Bowden's Hot Maxim Cover: '**30 Rock' Actress** Strips Down To Lingerie ..... '**30 Rock**' Finale: **Tina Fey**, Cast Play Two Lies And A Truth ...

"30 Rock" (2006) - Full cast and crew
www.imdb.com/title/tt0496424/fullcredits - Cached - Similar
30 Rock on IMDb: Movies, TV, Celebs, and more...

Figure 14: The first step in the instructions shown to the crowd workers while waiting for work to arrive

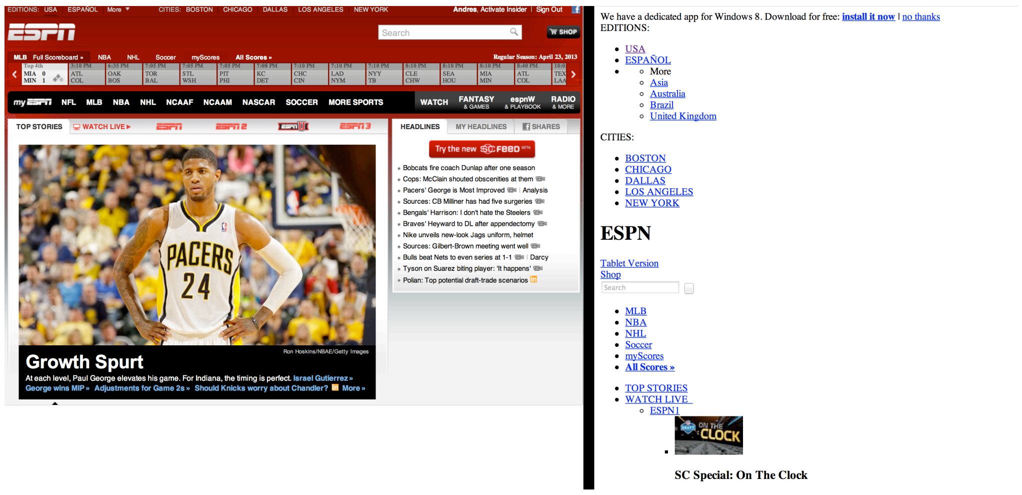
These instructions orient the worker while they wait, while also informing them that they are allowed to work on other tasks while wait. When an audio query is uploaded, the retainer system notifies the waiting workers that they should begin working. The task itself is described in detail in Section 3.2. The three recruited workers take their actions, either inputting new search terms or clicking on elements on the page. Since there are two options and three workers, there is consensus between the two options, providing quality control. This is especially important given that previous research shows that approximately one of three Turkers provide bad results [3].

As described in Section 3.2 and depicted in Figure 9, workers are allowed to click and vote on elements on the webpage associated with the query. When the workers load the page, there is a call to the server controller that retrieves the webpage associated with the query, but as a static page. This is then loaded in an iframe and shown to the workers. Getting the static page is done using the following algorithm, depicted in pseudocode below:

```
1. Get the source HTML of the page using the URL associated with the query
2. Make relative URLs absolute, using the base domain of the URL
3. Make Google search result links be direct links, not redirects
4. Make links inactive my changing "href" to "data-href"
5. Remove Javascript onclick attributes
6. Remove script elements
```

Because we have kept all CSS classes and element tags intact, the webpage has very similar affordances given to the worker. We also remove all Javascript, and break all links, so that any action the user takes will not change the contents of the webpage.

For some pages, removing Javascript and interactivity changes the affordances or makes salient information less so. On other pages, this completely changes the way the page looks, potentially confusing workers; for example in Figure 15 we compare espn.com rendered as a dynamic page and a static page. This is a limitation of our system, and we look to solve this in future work, described in Section 7.



- Bobcats fire coach Dunlap after one season
- Cops: McClain shouted obscenities at them
- Pacers' George is Most Improved | Analysis
- Sources: CB Milliner has had five surgeries
- Bengals' Harrison: I don't hate the Steelers
- Braves' Heyward to DL after appendectomy
- Nike unveils new-look Jags uniform, helmet
- Sources: Gilbert-Brown meeting went well
- Bulls beat Nets to even series at 1-1 | Darcy
- Tyson on Suarez biting player: 'It happens'
- Polian: Top potential draft-trade scenarios

We have a dedicated app for Windows 8. Download for free: install it now | no thanks
EDITIONS:

- USA
- ESPAÑOL
- More
  - Asia
  - Australia
  - Brazil
  - United Kingdom

CITIES:

- BOSTON
- CHICAGO
- DALLAS
- LOS ANGELES
- NEW YORK

ESPN

Tablet Version
Shop

- MLB
- NBA
- NHL
- Soccer
- myScores
- All Scores »

- TOP STORIES
- WATCH LIVE
  - ESPN1

SC Special: On The Clock

Figure 15: Espn.com rendered with (left) and without (right) Javascript

After the iframe loads this (now static) webpage, we add Javascript functions for hovering over and clicking on elements. The hovering function gives affordances, as shown in Figure 9, of what they will vote on. The clicking function adds a vote for that element. Once more than one worker has voted on the same element, it is sent to the server controller as an answer.

If the consensus is for inputting new search terms, then we have at least two sets of search terms. We use these terms as input to an algorithm that decides what the search terms should be. This algorithm uses Google's ASR API to get the text from the audio the current query. It then uses the following algorithm, depicted in pseudocode below, to calculate the best terms:

```
1. Stem (find root word for) all sets of search terms and the ASR text
2. Remove repeated words from each set of search terms and the ASR text
3. Initialize the score for each set of search terms to 0
4. For each word that is in more than one set of search terms, increase the score for
   that set of terms by 1
5. For each word that is in a set of search terms as well as the ASR text, increase the
   score for that set of terms by 0.5
6. The set of search terms with the higher score is chosen as the best, with ties chosen
   by the longer set of search terms
```

Workers are allowed to submit after one of three conditions are met. Entering search terms will enable submission. Agreeing with another worker on an element enables submission as long as the original worker has voted at least three times. Finally, voting on any element (even if it doesn't match another worker) enables submission after 85% of the allotted task work time is gone. This is to ensure we don't punish quality workers if the other two are not good.

Chapter 5

Driver User Study

This chapter describes the user study we performed to evaluate the FCB driver interface. We discuss the methodology of our experiment and the results.

Previous research from the NHTSA [19] indicates that interfaces showing non-driving critical information can be classified by risk according to several metrics. Tasks that can be done in steps involving less than 2 second glances are considered low risk, so we used this metric in our safety evaluation. The driver interface was also measured for ease of use, comfort, and confidence in the returned answer.

5.1 Methodology

In our user study, we focused on identifying usability problems and evaluating how effective and safe our system is. We recruited users by sending emails to the Computer Science department (undergrads, graduate students, and staff) at MIT. We requested current drivers at least 18 years old and who are fluent in English. We paid users $10 an hour, for up to two hours of time. We recruited a total of 10 users.

In our study, we measured the number and length of glances the driver had while using the two interfaces. In order to gather this data, we had two cameras filming the driver, one behind to capture interactions with the Android app, and one in front (using a computer's webcam) to capture the driver's glances. The setup is shown in Figure 16.

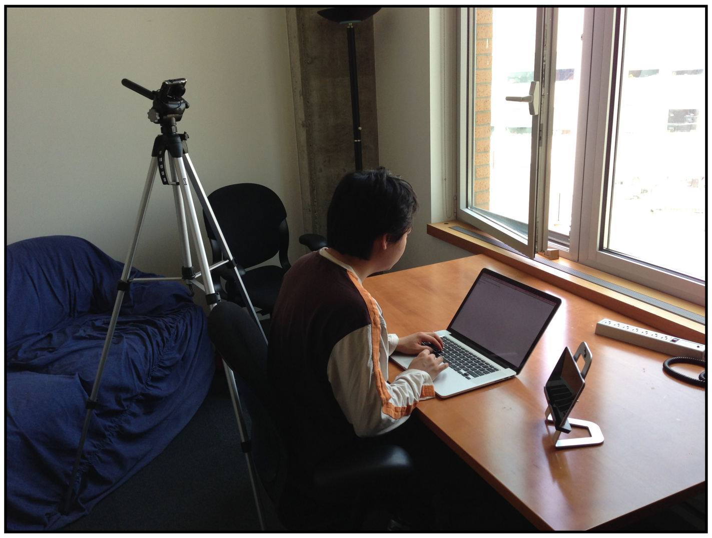
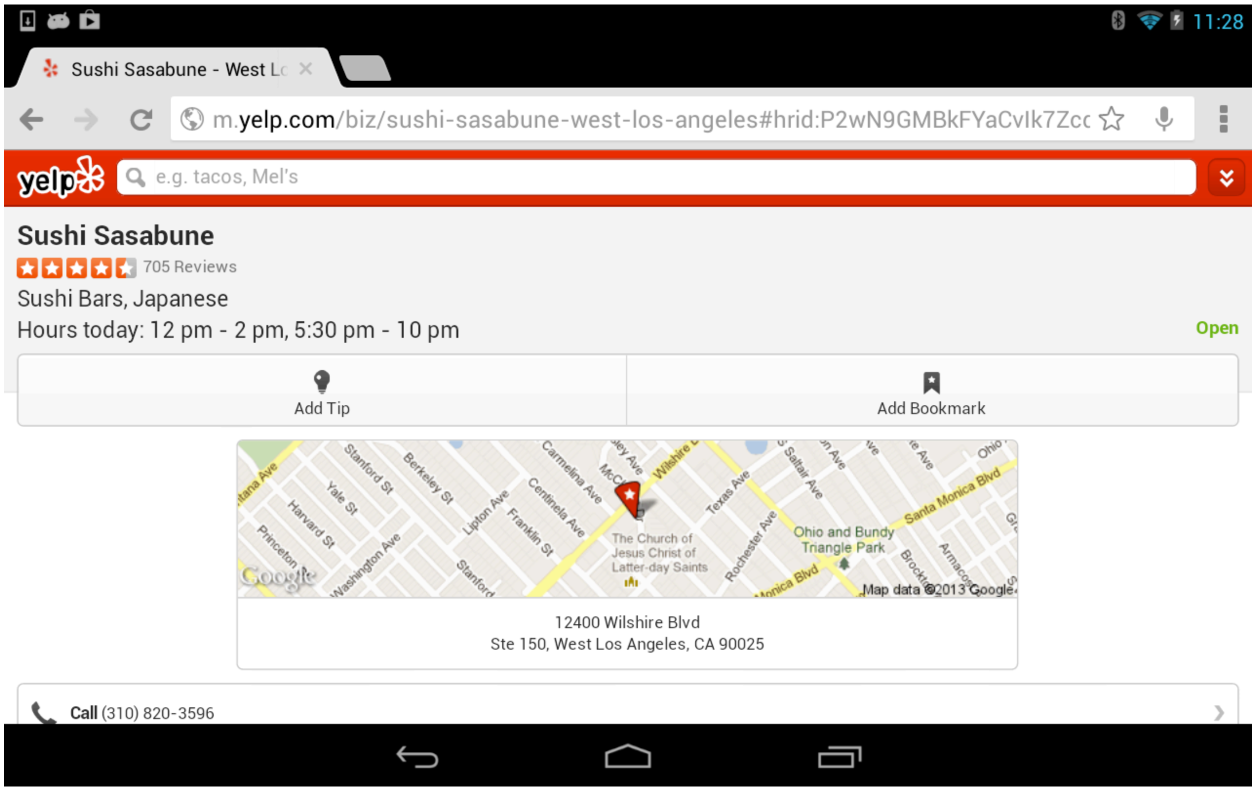


Figure 16: Experimental Setup: a camera from behind, a computer (with webcam and driving simulation) in front, and the Android tablet on the side, as if on a dashboard

We conducted a within-subject study in which each user used both FCB and the built-in Chrome browser on a Nexus 7 running Android 4.2.1. Before using either interface, we instructed the driver in how to use it. Our instructions pointed out key features they would be using. For Chrome, shown in Figure 17, this included how to initiate a new search, and how to use the ASR features and open new tabs. For FCB, we described that the system is powered by humans on the backend and instructed them what clicking on elements in the interface would do.


11:28

Sushi Sasabune - West Lc

m.yelp.com/biz/sushi-sasabune-west-los-angeles#hrid:P2wN9GMBkFYaCvIk7Zcc

e.g. tacos, Mel's

Sushi Sasabune

705 Reviews

Sushi Bars, Japanese

Hours today: 12 pm - 2 pm, 5:30 pm - 10 pm

Open

Add Tip

Add Bookmark

12400 Wilshire Blvd

Ste 150, West Los Angeles, CA 90025

Call (310) 820-3596

Wilshire
Santa Monica Blvd
The Church of Jesus Christ of Latter-day Saints
Ohio and Bundy Triangle Park
Map data ©2013 Google

Figure 17: Native Chrome web browser on Nexus 7, Android 4.2.1

For each interface, we gave them two queries and asked the user to come up with one of their own. The four queries we used are listed below; they were picked from the Mobile Needs examples [18]:

- “What is the phone number for Weight Watchers?”
- “What was the score of the NCAA final?”
- “What is in a mojito?”
- “What time does the Post Office close?”

The queries that the users came up with were good representations of questions one would ask while driving; most of them were asking to find an address, while some of them were trivia or point of interest information. Some examples are listed here:

- "Where is there swing dancing in Cambridge tonight?"
- "What is the height of the Green building?"
- "Where are there auto repair stations in Cambridge?"

Based on suggestions given in the NHTSA report [19] for testing the safety of driving interfaces, we used a low-fidelity driving simulation created by the New York Times [8], as shown in Figure 18. This allowed us to measure the distraction caused by using the interface while performing the driving simulation, which the NHTSA has said is a sufficient metric to gauge safety [19].

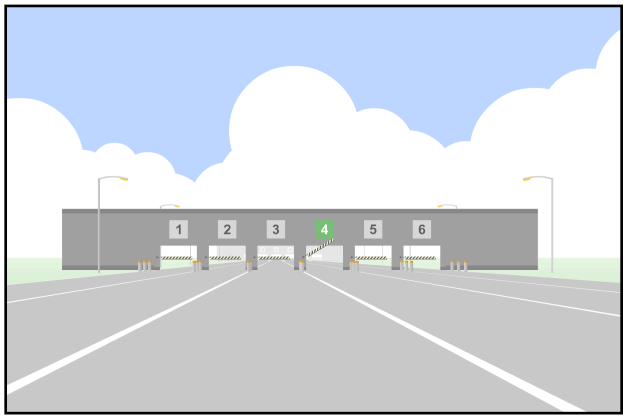


Figure 18: A driving simulation by the NYT that asks users to press numbers on their keyboard to go through gates

The order of the interfaces and queries were counterbalanced to ensure that we weren't biasing the data by choosing queries which were easier to answer on one interface. The worker results were provided using the "Wizard of Oz" method. This meant we were acting as the three crowd workers, simulating both the accuracy and speed in which they return answers. This was done in order to ensure consistency between simulations.

After finishing each query, the user was asked to fill out a brief survey, rating on a scale of 1-7 the following three questions: "how easy was it to find the information for each query?", "how comfortable were you while using the application?", and "how confident were you that the answer you found was accurate?". We also asked them for any other, non-quantitative, feedback they wanted to give to help instruct next iterations of FCB.

5.2 Results

After gathering the data for our study, we analyzed it in several ways to measure objective safety as well as subjective preferences between the two interfaces. In order to measure the length and count of glances, we watched the video recorded of the driver's face and measured each time he looked away from the screen.

We found that for FCB, there were a total of 945 glances, averaging 0.8 seconds in length. For Chrome, there were a total of 929 glances, averaging 1.3 seconds in length. According to the NHTSA [19], glances that are longer than 2 seconds are extremely dangerous, and we found that for FCB 1.5% of the glances were of that length, compared to 14.2% of glances for Chrome. A histogram of glance lengths is shown in Figure 19, with the means graphed as well.

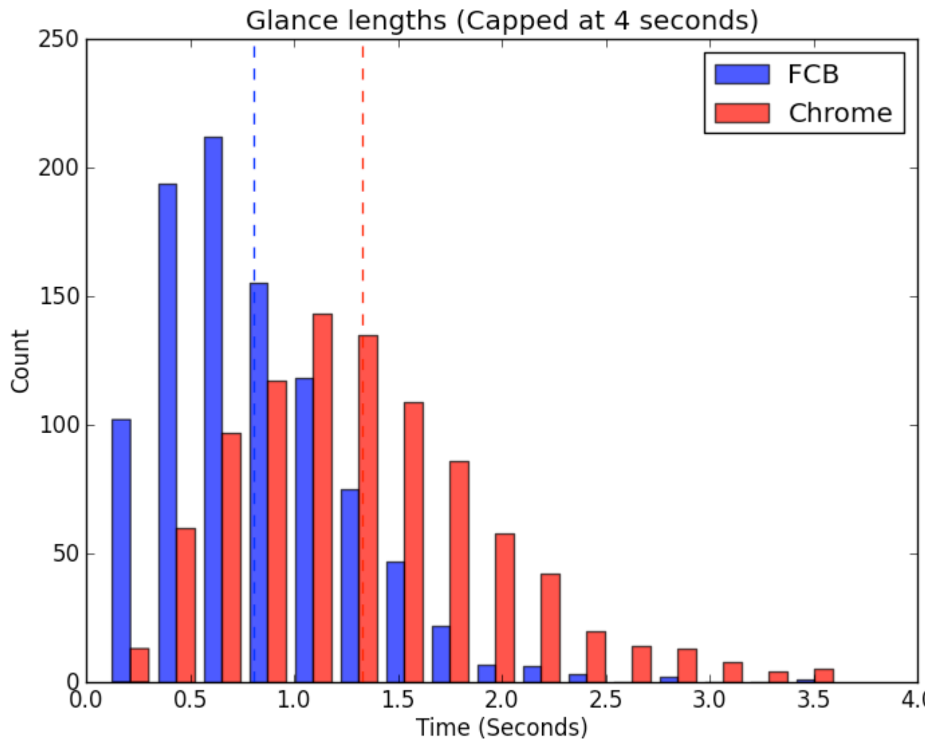


Figure 19: Histogram of glance lengths (for < 4.0 seconds). Vertical lines indicate mean for two interfaces

The difference between the glance lengths was compared using an unpaired t-test on the log transformed data, and we found strong significance ($t = 21.5657$, $df = 1872$, p-value < 0.0001), showing that users glanced away for longer when using Chrome when compared to FCB. Many of the longer glances while using Chrome happened when the user was reading a paragraph of information to pick out key points or when the user was scrolling through several pages to find where on the page the answer was actually located. In FCB, that work was done by the workers. Additionally, some users got frustrated with Chrome's voice recognition failures and resorted to typing in order to ensure their query was written correctly. This often was not needed, as slightly incorrect queries are automatically corrected during a Google search. Again, workers generated search terms for FCB, and the driver was never allowed to enter text.

Although the length of glances were much shorter in FCB than in Chrome, the count was very similar, which is unsafe [19]. Many users glanced at the screen even when nothing was changing to see if any progress was being made. These glances could be removed by adding more audio feedback so the driver can hear what the workers are doing and are assured the task is progressing.

We also gathered subjective data from each user. Graphed in Figure 20 are the averages of the ratings for each question and each interface. The error bars indicate the standard error.

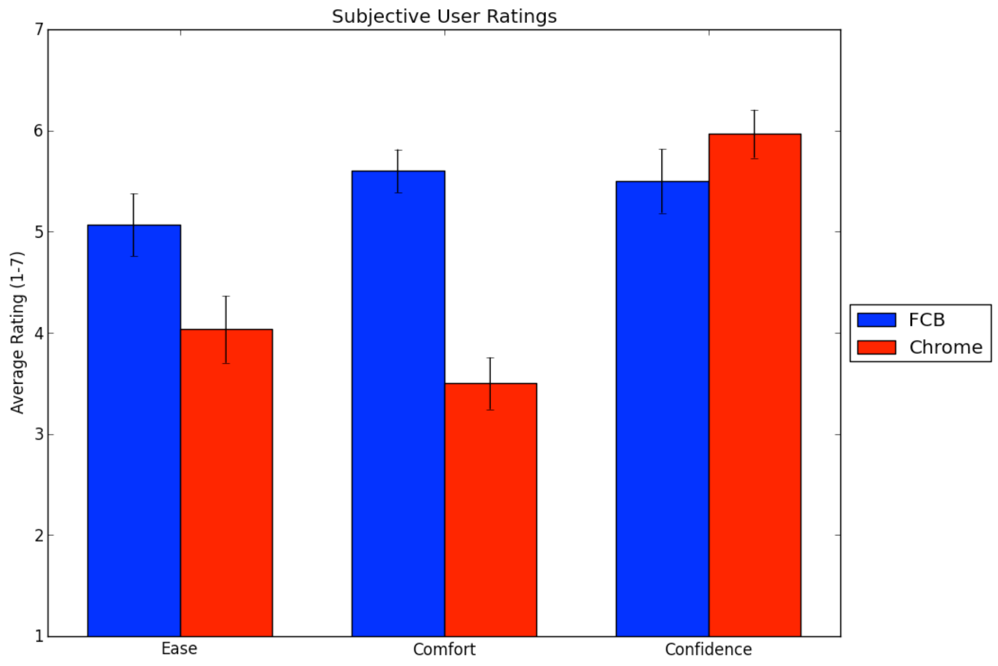


Figure 20: Average of answer to subjective questions. Standard error is shown as error bars. Ratings were on a 1-7 scale, with 7 being the highest positive rating.

These averages indicate that the ease of use and comfort were higher for FCB, but that users were more confident in the answers found using Chrome. It was a within-subject study, so we ran three paired t-tests in order to determine significance. We found that ease of use was not significant (t = -1.9438, df = 9, p-value = 0.08379), but comfort was (t = -5.4537, df = 9, p-value = 0.0004037). This means that while users didn't necessarily think finding information on FCB was easier, they were more comfortable using it while driving. For confidence in the correctness of the answer, we found that there was no significant difference (t = 0.7397, df = 9, p-value = 0.4783). This suggests that we are providing enough contextual information in FCB that the user is as confident in an answer found by workers versus an answer found by themselves.

It was clear from our study that Chrome on the tablet was not intended to be used by mobile users. Many buttons, including the ASR microphone, had very small hit targets that caused frustration for users. This was confirmed by their qualitative feedback. One user noted that "the text input and speech input [on Chrome] don't play well together." Additionally, as we've seen in previous studies [11], having inertial scrolling in an interface greatly increases risk while driving, and Chrome is built on scrolling, not discrete screens of data like FCB. Therefore, we find that driving interfaces need to be custom built to ensure safety, as common methods of interaction for stationary use are not appropriate for mobile use.

One major difference between drivers using Chrome and FCB was the phrasing of their query. As one user stated, "I found myself wanting to phrase queries in complete sentences or phrases I would not normally search for on Google. I wanted to use natural speech." This indicates that users expect FCB to be more end-to-end, and their interactions with it are to guide it along or correct missteps. On the other hand, many drivers using Chrome walked the browser through the interaction step-by-step, for instance making a Google search for “Wikipedia May 1st” instead of the high-level query “Is May 1st a holiday?” However, because of the nature of the two interfaces, it was much harder for users to correctly determine if an answer was complete or not using FCB. For instance, when searching for “What is in a mojito?” users often got incomplete lists (usually just three ingredients due to the number of results shown on the interface), but they were unaware that there should be more. On Chrome, this would be immediately obvious, as they can see that the ingredients all listed together.

Another important aspect of web browsing is modifying your query when you get slightly incorrect results. We saw this on some queries where the user made a query for “find me the nearest Chinese restaurant”. The results showed several names of Chinese restaurants nearby, but the user just wanted the address. On Chrome, this was often as simple as looking further on the page. On FCB, users did the correct thing and spoke into the mic again, modifying their query to “find me the address of the nearest one”. This shows that users intuitively know that they are allowed to modify a query as if they are talking to a human, not trying to use a screen reader or direct a search engine.

Chapter 6

Crowd User Study

This chapter describes the user study we performed to evaluate the crowd worker interface. We discuss the methodology of our experiment and the results.

6.1 Methodology

In our user study for the crowd interface, we wanted to measure the accuracy and speed in which workers returned answers. We used twelve queries, listed below, and each was designated as hard or easy. Hard queries have answers that depend on the driver's location or are specific to a date or time. For each query, we had three different webpages, giving us 36 tasks, where a task is a query combined with a webpage. We used three different websites in order to cover the three types of interactions the workers can choose to do: click on the answer directly, start a new search with new search terms, and click on links that will take the user to a new page.

Types of webpages presented to workers:

- Answer is located on the page, such as a Wikipedia page
- A completely unrelated webpage, which would need a new Google search
- A Google results page with the correct search terms already entered, with links to the answer or the answer directly

Easy worker queries:

- "What does Tim Cook look like?"
- "How much does an iPhone 5 cost?"
- "Where is Passion Pit from?"
- "What is the phone number for weight watchers?"
- "When is the Mika Concert?"

- “What is in a mojito?”

Hard worker queries:

- “Where is the nearest bookstore?”
- “What was the score of the NCAA Final?”
- “What time does the post office close?”
- “Where is the World Cup going to be?”
- “Where is Evil Dead playing tonight?”
- “How long does it take to get to Philadelphia?”

To simplify evaluation, we collected work from individual workers for each task, and stored their actions along a timeline. We gathered ~18 workers’ timelines for each task. With this data, we generated ~816 trials for each task, where a trial is a unique combination of three workers’ timelines. From this, we can simulate synchronous work by having each worker’s timeline start at the same time and running the same analysis as described in Section 3.2 to gather results. This simulates the timing and output of responses that would have been gathered by workers acting synchronously. We used Mechanical Turk [1] workers, and paid them 5 cents per task completed. They were given 11 minutes to complete it, but most tasks were easily accomplishable in a minute or less. We ignored any actions past 100 seconds, as that latency isn’t acceptable for drivers, and those workers would be rejected. After gathering results, we manually marked each one as accurate or inaccurate. Each result was only deemed correct if it led the driver to a new page closer to the result or if it returned the answer to the driver.

6.2 Results

6.2.1 Correctness in Classification of Task

We first determined if workers agreed on the correct type of voting to do: entering search terms or clicking on elements. A cumulative histogram of the correctness of this step is shown in Figure 21.

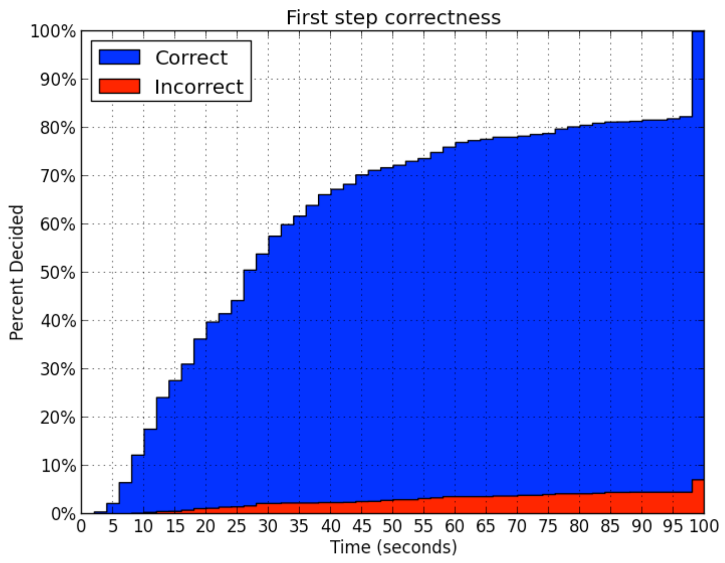


Figure 21: Percent of trials correct in deciding between voting on search terms or click voting

Workers answered most questions with the correct type of action: either new search terms or clicking on elements. In 30 seconds from the start of the task, ~56% of tasks were correctly classified and only ~2% of them were incorrectly classified, with ~42% still undecided. After 90 seconds, ~77% were classified correctly and ~5% were wrong, with ~18% still undecided. Considering the assumption that one in three Turker is unreliable [3], this is a large improvement, but that comes with an increase in latency.

6.2.2 Correctness for Click Voting

In order to evaluate the full correctness of a task, we also need to incorporate the quality of the answer returned, which compounds with the errors in Section 6.2.1. For the tasks where the answer was on the page, and thus workers should be taking the action of clicking on elements, we had to consider two types of drivers: a driver who trusts the first result and a driver that waits for all three results to show up. The results for these are shown in Figure 22 and Figure 23, respectively.

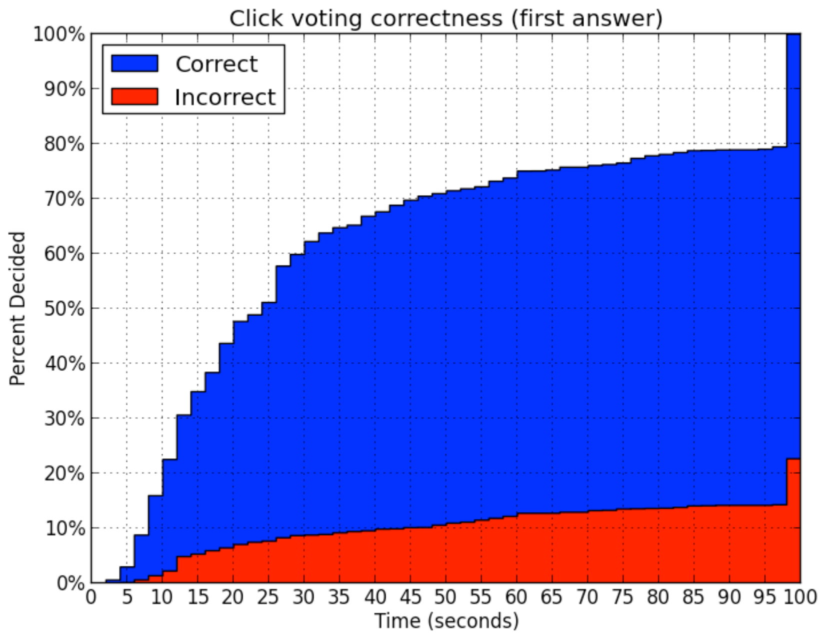


Figure 22: Percent of trials where the answer was on the page and the first returned result answered the query

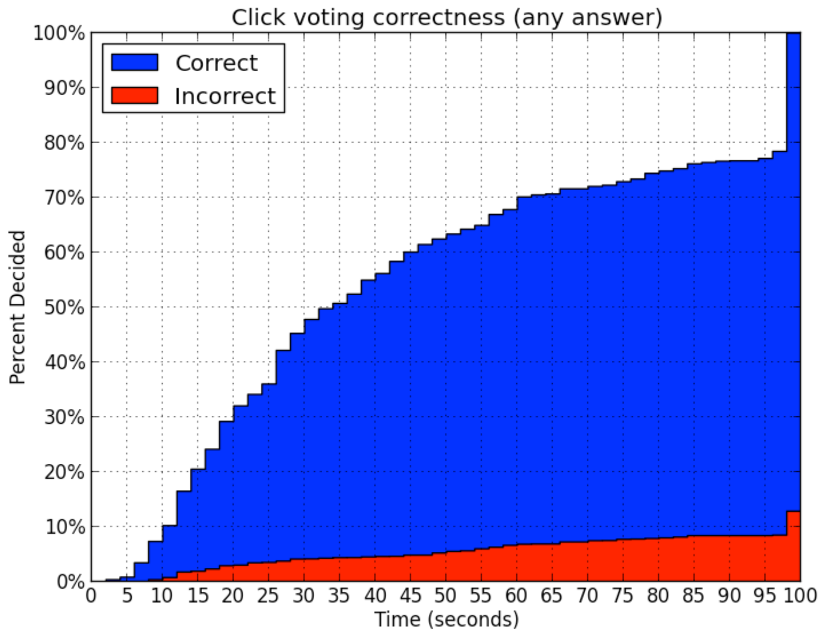


Figure 23: Percent of trials where the answer was on the page and any of the three returned results answered the query (time is based on the last returned result)

Note that these graphs compound the errors from Figure 21; if the task had the answer on the page and users decided to submit search terms instead, they will show up in Figure 22 and Figure 23 as incorrect.

As would be expected, accuracy is improved for drivers that wait to see all three results. In 30 seconds, ~44% of tasks had three results and at least one was correct, ~4% had three results and none were correct, and ~52% had less than three results. Comparatively, for drivers that only went off the first returned result, ~60% were correct, ~9% were incorrect, and ~31% did not have any result yet. This again shows that our click voting system is an improvement on systems without quality control; within 30 seconds we have ~13% answers returned incorrect (compared to ~33% for a single, unreliable, Turker). However, this is at the cost of an increase in latency, since we are waiting for agreement between multiple workers, which takes longer than just returning results from a single worker.

Many of the incorrect results for click voting were because there were many salient HTML elements at the top of the page related to the query which didn't show the answer. Part of our future work is finding criteria for disabling elements that we know will not have the answer in them.

6.2.3 Correctness for Search Terms

The second type of task are those which need new search terms. As opposed to click voting, the results varied on whether the query was defined as easy or hard. These correctness graphs are shown in Figure 24 and Figure 25.

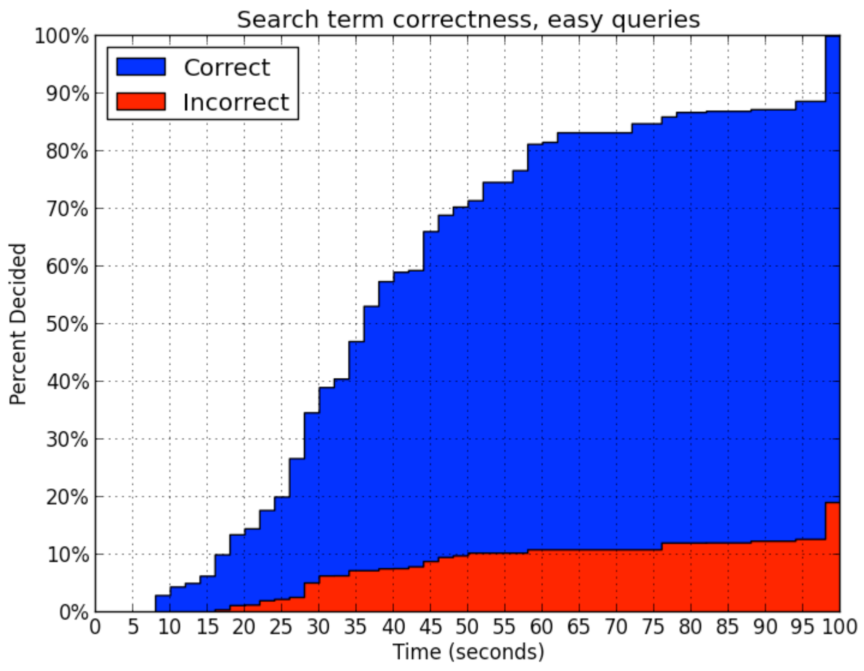


Figure 24: Percent of trials for easy queries that received correct new search terms

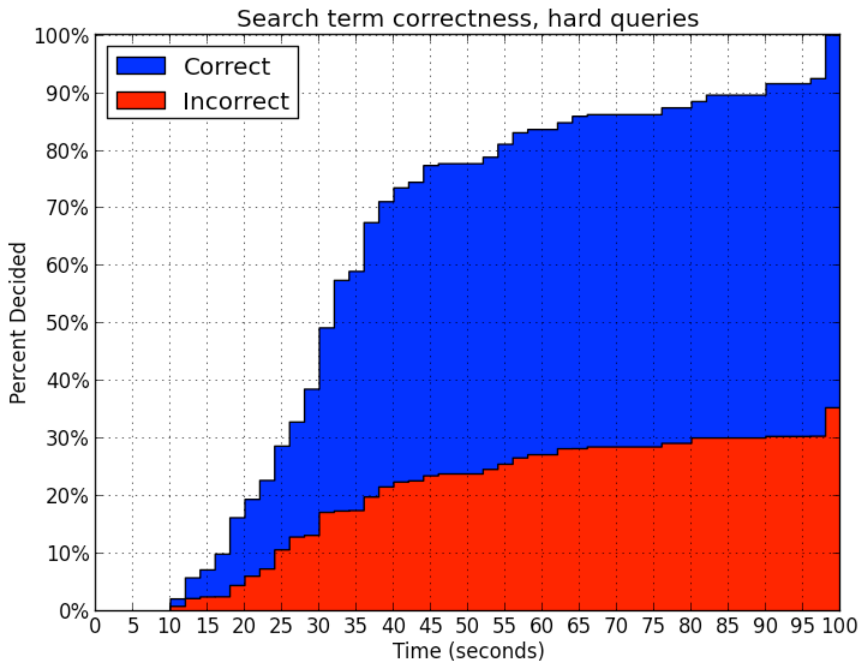


Figure 25: Percent of trials for hard queries that received correct new search terms

It took workers longer to generate search terms than to perform click voting and it was less likely to be correct, as they needed to actually type in words, often transcribing the driver's query. After 30 seconds for easy tasks which required search terms, ~43% of workers had agreed on correct search terms, ~6% agreed on incorrect terms, and ~51% were still undecided. Similarly, after 30 seconds for hard tasks, ~31% of workers had agreed on correct search terms, ~28% agreed on incorrect terms, and ~41% were undecided. Most of the incorrect search terms for hard queries didn't include the driver's location or the requisite time/date information to get correct results. For both types of queries, our results are still better than using a single, unreliable worker.

Chapter 7

Future Work and Conclusion

7.1 Driver Interface

Our study shows that FCB is safer than Chrome, but there are still many aspects that can be improved. Because of limitations in the text-to-speech system in Android, we need to develop our own method of scraping text to improve how it is read aloud. This includes stripping citations from text and formatting date and time strings so they make sense when read aloud, as those were confusing and frustrating to drivers when output incorrectly.

In order to fully follow the NHTSA guidelines for safety [19], we need to decrease the total number of glances when using FCB. Adding additional audio cues would help reduce the number of needless glances away from the road just to confirm that nothing is changing on screen. Reducing the latency of the crowd interface would also decrease the total number of glances away from the screen.

7.2 Crowd Interface

Many of the problems with the crowd interface stemmed from needing to add location data, since a large number of questions are about local points of interest or driving destinations. We plan on including location cookies in future work so the workers' search results are local to where the driver is unless overridden by the search terms. This matches how drivers and workers would expect their searches to work. We also plan on including autocompletion, which would improve the speed and accuracy in which workers can type out queries, especially for proper nouns and full phrases.

As was shown in Figure 15, issues can arise when webpages depend on Javascript or Flash. This can be solved in future work by having workers work in a virtual machine where we can better control the browser they are using and prevent them from changing pages, while still allowing Javascript and other dynamic elements on webpages.

7.3 Overall System

There are many ways this system can be generalized and extended to further enhance the web browsing experience while driving. Our scope does not fully cover the actions users take on the web; primarily it excludes personalized information needs. This includes personal communication such as Facebook, Google Chat, and email. In order to solve these needs, the system would need to sanitize the user's information in order to obscure it from workers. Part of this extension would also include interactions to instruct workers to input data into forms, which provides new challenges to ensure the driver has agency over what is being typed on sites as well as keeping safety as a priority.

Another possible extension is to allow different types of crowds. These could include friends in your social circle or other types of paid crowd workers. This could allow for more personalized interactions or longer threads of conversation, more closely mirroring the proxy browsing that we strove to emulate.

7.4 Conclusion

This project is a promising beginning to a web browser that is safe for drivers. There are many ways to extend FCB to better accommodate the needs of drivers, but based on our findings, the current level of interactivity is desired by users, and certainly much safer than the built-in alternative.

References

1. "Amazon's Mechanical Turk." Web. <www.mturk.com>
2. Bernstein, Michael; Brandt, Joel; Miller, Robert; and Karger, David. "Crowds in Two Seconds: Enabling Realtime Crowd-Powered Interfaces." UIST 2011. ACM 978-1-4503-0716-1/11/10.
3. Bernstein, Michael S.; Little, Greg; Miller, Robert C.; Hartmann, Björn; Acherman, Mark S.; Karger, David R.; Crowell, David; Panovich, Katrina. "Soylent: A Word Processor with a Crowd Inside." UIST 2010. ACM 978-1-4503-0271-5/10/10.
4. Bigham, Jeffrey; Jayant, Chandrika; Ji, Hanjie; Little, Greg; Miller, Andrew; Miller, Robert; Miller, Robin;Tatarowicz, Aubrey; White, Brandyn; White, Samuel; Yeh, Tom. "VizWiz: Nearly Real-time Answers to Visual Questions." UIST 2010. ACM 978-1-60558-745-5/09/10.
5. Bila, Nilton; Ronda, Troy; Mohomed, Iqbal; Truong, Khai; and de Lara, Eyal. "PageTailor: Reusable End-User Customization for the Mobile Web." MobiSys 2007. ACM 978-1-59593-614-1/07/0006.
6. Borodin, Yevgen. "Bridging the Web Accessibility Divide." Stony Brook University PHD Dissertation. December 2009.
7. Chelba, Ciprian; Bikel, Dan; Shugrina, Maria; Nguyen, Patrick; Kumar, Shankar. "Large Scale Language Modeling in Automatic Speech Recognition." Google, Inc.
8. Dance, Gabriel; Jackson, Tom; Pilhofer, Aron. "Gauging Your Distraction." New York Times. August 4th, 2009.
9. "Google Now." Web. <http://www.google.com/landing/now/>
10. Kumar, Ranjitha; Talton, Jerry;Ahmad, Salman; Klemmer, Scott. "Bricolage: Example-Based Retargeting for Web Design." CHI 2011. ACM 978-1-4503-0267-8/11/05.
11. Lasch, Annegret and Kujala, Tuomo. "Designing Browsing for In-Car Music Player - Effects of Touch Screen Scrolling Techniques, Items Per Page and Screen Orientation on Driver Distraction." AutomotiveUI 2012. ACM 978-1-4503-1751-1/12/10.

12. Law, Edith and Zhang, Haoqi. “Towards Large-Scale Collaborative Planning: Answering High-Level Search Queries Using Human Computation.” AAAI Conference on Artificial Intelligence 2011.
13. Matuszek, Cynthia; Herbst, Evan; Zettlemoyer, Luke; Fox, Dieter. “Learning to Parse Natural Language Commands to a Robot Control System.” University of Washington, Seattle.
14. Morris, Meredith; Teevan, Jaime; and Bush, Steve. “Enhancing Collaborative Web Search with Personalization: Groupization, Smart Splitting, and Group Hit-Highlighting.” CSCW 2008. ACM 978-1-60558-007-4/08/11.
15. Nayak, Rajeev. “Sinch: Searching Intelligently on a Mobile Device.” MIT M.Eng. Thesis. August 20th, 2010.
16. Reimer, Bryan; Mehler, Bruce; Wang, Ying; Mehler, Alea; McAnulty, Hale; Mckissick, Erin; Coughlin, Joseph; Matteson, Steve; Levantovsky, Vladimir; Gould, David; Chahine, Nadine; Greve, Geoff. “An Exploratory Study on the Impact of Typeface Design in a Text Rich User Interface on Off-Road Glance Behavior.” AutomotiveUI 2012. ACM 978-1-4503-1751-1/12/10.ed
17. “Siri.” Web. <http://www.apple.com/ios/siri/>
18. Sohn, Timothy; Li, Kevin; Griswold, William; Hollan, James. “A Diary Study of Mobile Information Needs.” CHI 2008. ACM 978-1-60558-745-5/09/10.
19. Strickland, David, et. al. “Visual-Manual NHTSA Driver Distraction Guidelines for In-Vehicle Electronic Devices.” Docket No. NHTSA-2010-0053.
20. Tellex, Stefanie; Thaker, Pratiksha; Beits, Robin; Simeonov, Dimitar; Kollar, Thomas; Roy, Nicholas. “Toward Information Theoretic Human-Robot Dialog.”
21. Terken, Jacques; Visser, Henk-Jan; Tokmakoff, Andrew. “Effects of Speech-based vs Handheld E-mailing and Texting on Driving Performance and Experience.” AutomotiveUI 2011. ACM 978-1-4503-1231-8/11/11.
22. “VoiceOver.” Web. <http://www.apple.com/accessibility/voiceover/>
23. von Ahn, Luis and Dabbish, Laura. “Labeling Images with a Computer Game.” CHI 2004, ACM 1-58113-702-8/04/0004.
24. Whang, Yonghyun; Whang Yonghyun; Jung, Changwoo; Kim, Jihong; and Chung, Sungkwon. “WebAlchemist: A Web Transcoding System for Mobile Web Access in Handheld Devices.” SPIE 2001 Proceedings; Conference Volume 4534.

25.Wiltse, Heather and Nichols, Jeffrey. “PlayByPlay: Collaborative Web Browsing for Desktop and Mobile Devices.” CHI 2009. ACM 978-1-60558-246-7/09/04.