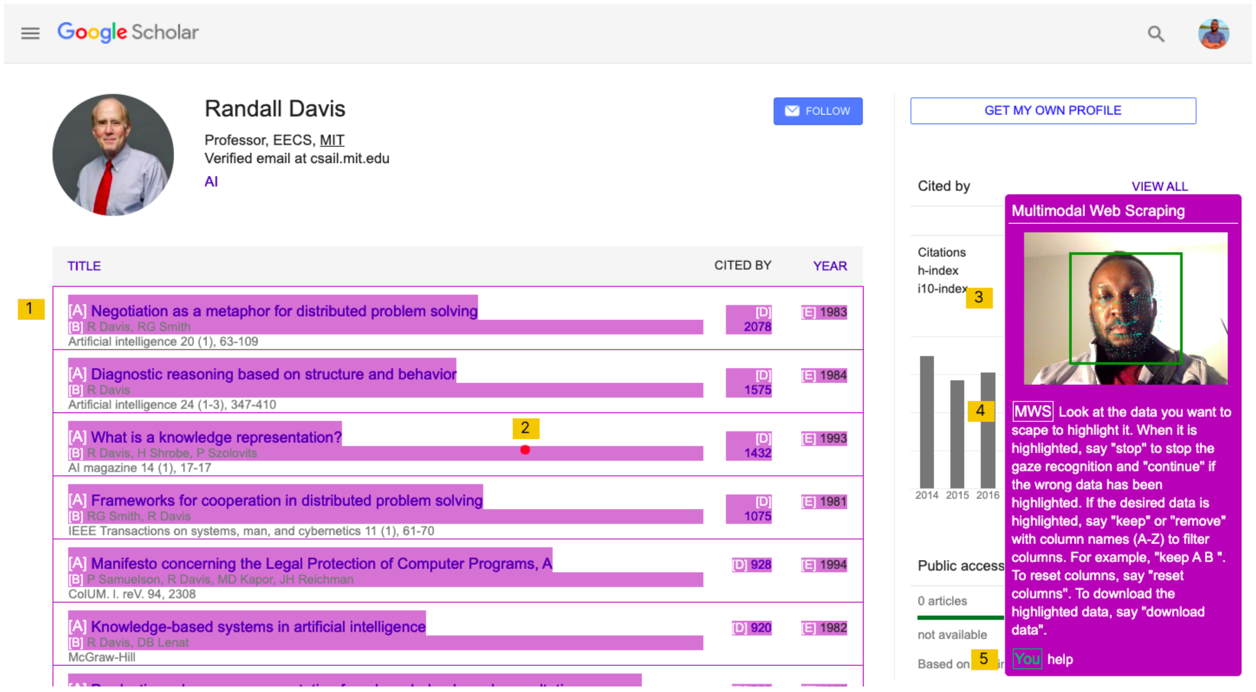
# MWS: Multimodal Web Scraping



Randall Davis

Professor, EECS, MIT
Verified email at csail.mit.edu
AI

| TITLE | CITED BY | YEAR |
|---|---|---|
| [A] Negotiation as a metaphor for distributed problem solving<br>[B] R Davis, RG Smith<br>Artificial intelligence 20 (1), 63-109 | [D] 2078 | [E] 1983 |
| [A] Diagnostic reasoning based on structure and behavior<br>[B] R Davis<br>Artificial intelligence 24 (1-3), 347-410 | [D] 1575 | [E] 1984 |
| [A] What is a knowledge representation?<br>[B] R Davis, H Shrobe, P Szolovits<br>AI magazine 14 (1), 17-17 | [D] 1432 | [E] 1993 |
| [A] Frameworks for cooperation in distributed problem solving<br>[B] RG Smith, R Davis<br>IEEE Transactions on systems, man, and cybernetics 11 (1), 61-70 | [D] 1075 | [E] 1981 |
| [A] Manifesto concerning the Legal Protection of Computer Programs, A<br>[B] P Samuelson, R Davis, MD Kapor, JH Reichman<br>ColUM. I. reV. 94, 2308 | [D] 928 | [E] 1994 |
| [A] Knowledge-based systems in artificial intelligence<br>[B] R Davis, DB Lenat<br>McGraw-Hill | [D] 920 | [E] 1982 |

MWS Look at the data you want to scape to highlight it. When it is highlighted, say "stop" to stop the gaze recognition and "continue" if the wrong data has been highlighted. If the desired data is highlighted, say "keep" or "remove" with column names (A-Z) to filter columns. For example, "keep A B ". To reset columns, say "reset columns". To download the highlighted data, say "download data".

*Overview of how MWS can be used to scrape data from a website: 1) Highlighted website data corresponding to the scraped data table, 2) red circle which represents the user's detected  gaze, 3) webcam feed showing how user's gaze is being detected, 4) MWS feedback dialog (also uttered via speech synthesis) used to respond to user commands and guide user actions, 5) user feedback dialog used to display what voice command was detected by the system*

MWS is a Chrome browser extension that enables web scraping via gaze detection and voice commands in a few simple steps. Once initiated on a website, MWS detects the point on the website where the user is looking, through their webcam feed, and continuously determines and highlights the data table that that point belongs to. When the desired data table is highlighted, a user can use voice commands to stop the gaze detection and save the highlighted data table, filter which of the data table's columns to keep or remove and download the data table as a JSON file.

Team Members: Kapaya Katongo

Paper: https://github.com/Kapaya/multimodal-web-scraping/blob/main/paper.pdf

Demo: [https://youtu.be/Ee34bGjcYgU](https://youtu.be/Ee34bGjcYgU)

Code: https://github.com/Kapaya/multimodal-web-scraping

## INTRODUCTION AND OVERVIEW

### Motivation

Web scraping is the process of extracting data from a website. It can be performed via traditional programming by writing code or via *programming-by-example* by providing examples of the desired data (through mouse selections, clicks etc) to a system which generalizes the examples to the rest of the data set.

Programming-by-example-based web scraping uses a technique called *wrapper induction* to generalize from a few user provided examples of the desired data to the entire data set. Here, “wrapper” refers to the program that determines the desired data table and “induction” refers to the process of generating a wrapper for a given website (websites are laid out differently and so will need different wrappers).

The primary purpose of user examples in programming-by-example-based web scraping  is to determine the user’s intent. Is there a more natural way to determine user intent that combines multiple modalities?

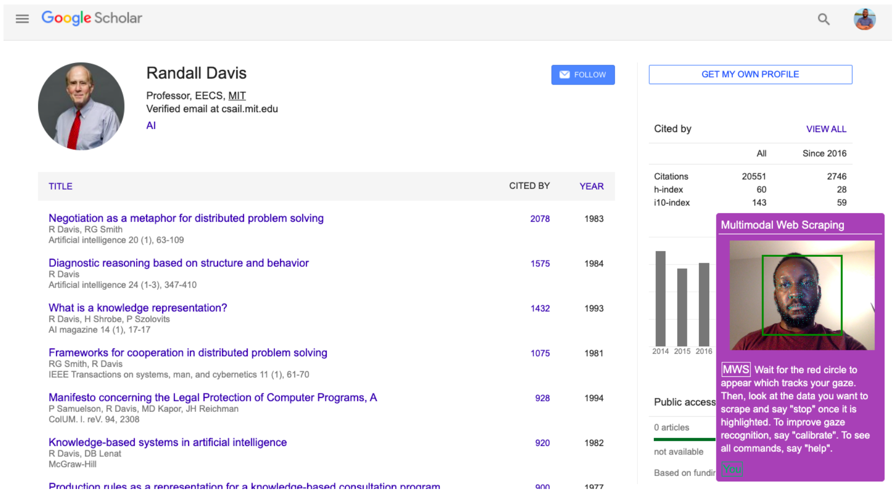
### Solution

MWS, short for Multimodal Web Scraping, is a Chrome browser extension that enables web scraping via gaze detection and voice commands. The key insight is that if a user wants to scrape data on a website, they are likely going to look at it. Therefore, a user’s gaze can be used to determine their intent in a more natural manner than requiring them to explicitly provide examples via mouse interactions. To give users control over the scraping process, MWS provides a voice interface which, again, provides a more natural user interaction than mouse interactions.

## SYSTEM DESCRIPTION

### User Scenario

After viewing Professor Randall Davis’s publications on Google Scholar, Kapaya wants to scrape the publication data to perform some analysis. Using MWS, he can scrape and download the data as a JSON file using a few simple steps without having to leave his browser or write code. He starts out by initiating MWS his browser’s context menu which renders its interface shown below:



Randall Davis
Professor, EECS, MIT
Verified email at csail.mit.edu
AI

| TITLE | CITED BY | YEAR |
|---|---|---|
| Negotiation as a metaphor for distributed problem solving<br>R Davis, RG Smith<br>Artificial intelligence 20 (1), 63-109 | 2078 | 1983 |
| Diagnostic reasoning based on structure and behavior<br>R Davis<br>Artificial intelligence 24 (1-3), 347-410 | 1575 | 1984 |
| What is a knowledge representation?<br>R Davis, H Shrobe, P Szolovits<br>AI magazine 14 (1), 17-17 | 1432 | 1993 |
| Frameworks for cooperation in distributed problem solving<br>RG Smith, R Davis<br>IEEE Transactions on systems, man, and cybernetics 11 (1), 61-70 | 1075 | 1981 |
| Manifesto concerning the Legal Protection of Computer Programs, A<br>P Samuelson, R Davis, MD Kapor, JH Reichman<br>ColUM. l. reV. 94, 2308 | 928 | 1994 |
| Knowledge-based systems in artificial intelligence<br>R Davis, DB Lenat<br>McGraw-Hill | 920 | 1982 |
| Production rules as a representation for a knowledge-based consultation program | 900 | 1977 |

Cited by

|  | All | Since 2016 |
|---|---|---|
| Citations | 20551 | 2746 |
| h-index | 60 | 28 |
| i10-index | 143 | 59 |

Multimodal Web Scraping

MWS Wait for the red circle to appear which tracks your gaze. Then, look at the data you want to scrape and say "stop" once it is highlighted. To improve gaze recognition, say "calibrate". To see all commands, say "help".

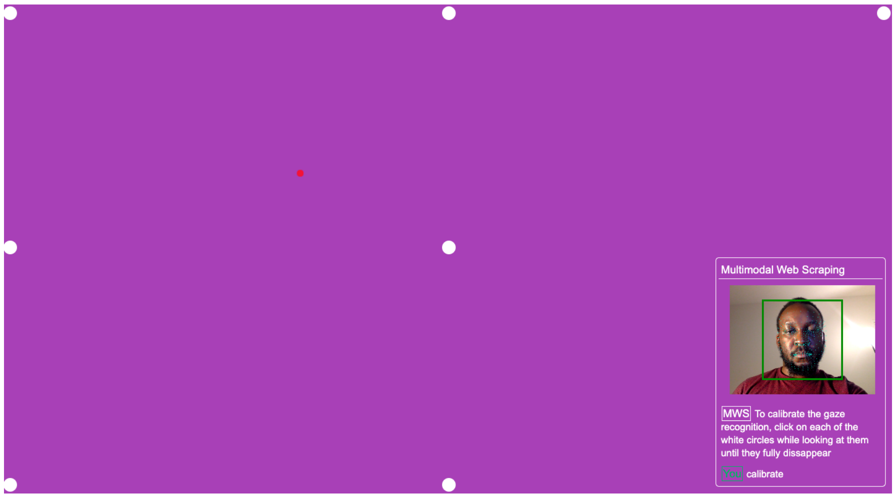
You

The interface consists of a dialog through which Kapaya can communicate with MWS and a webcam feed that shows him how to position himself for the gaze detection. Upon opening, MWS speaks out the welcome message in its dialog. The message provides an overview of how to use MWS and access the list of available voice commands.

## Calibrating Gaze Detection

Since this is the first time MWS is being run on Google Scholar in Kapaya's browser, the gaze detection has poor accuracy. As instructed in the welcome message, Kapaya says "calibrate" in order to calibrate MWS. Subsequent uses on Google Scholar will not require calibration as the data will be stored in the browser's local storage.

The gaze detection calibration is done right in the website as shown below:



Multimodal Web Scraping

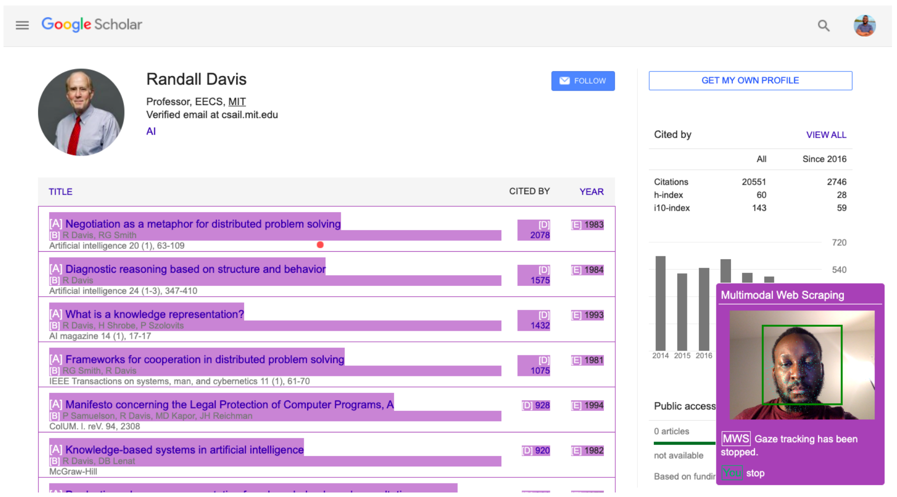
MWS To calibrate the gaze recognition, click on each of the white circles while looking at them until they fully dissappear

You calibrate

To calibrate MWS, Kapaya simply clicks on each of the white circles while looking at them until they disappear. When the final circle disappears, the calibration process automatically ends.

## Selecting Data To Scrape Via Gaze Detection

Now that MWS has been calibrated and the accuracy of the gaze detection has improved, Kapaya can proceed with the scraping process. As he looks around the website, a red circle tracks his gaze and highlights the data table that the point under the red circle belongs to. The highlighted data table includes column names as single, capitalized letters in square brackets next to each column value. To highlight the publication data, Ben simply looks at one of the publications and says "stop" to stop the gaze detection and save the currently highlighted data:



Google Scholar

Randall Davis
Professor, EECS, MIT
Verified email at csail.mit.edu
AI

| TITLE | CITED BY | YEAR |
|---|---|---|
| [A] Negotiation as a metaphor for distributed problem solving<br>[B] R Davis, RG Smith<br>Artificial intelligence 20 (1), 63-109 | [D] 2078 | [E] 1983 |
| [A] Diagnostic reasoning based on structure and behavior<br>[B] R Davis<br>Artificial intelligence 24 (1-3), 347-410 | [D] 1575 | [E] 1984 |
| [A] What is a knowledge representation?<br>[B] R Davis, H Shrobe, P Szolovits<br>AI magazine 14 (1), 17-17 | [D] 1432 | [E] 1993 |
| [A] Frameworks for cooperation in distributed problem solving<br>[B] RG Smith, R Davis<br>IEEE Transactions on systems, man, and cybernetics 11 (1), 61-70 | [D] 1075 | [E] 1981 |
| [A] Manifesto concerning the Legal Protection of Computer Programs, A<br>[B] P Samuelson, R Davis, MD Kapor, JH Reichman<br>ColUM. l. reV. 94, 2308 | [D] 928 | [E] 1994 |
| [A] Knowledge-based systems in artificial intelligence<br>[B] R Davis, DB Lenat<br>McGraw-Hill | [D] 920 | [E] 1982 |

| Cited by | All | Since 2016 |
|---|---|---|
| Citations | 20551 | 2746 |
| h-index | 60 | 28 |
| i10-index | 143 | 59 |

Multimodal Web Scraping

MWS Gaze tracking has been stopped.

You stop

## Filtering Columns To Scrape In Selected Data

Now that the publication data Ben wants scrape is highlighted and saved, he can decide which columns to keep or remove through voice commands. Since he is only interested in the publication title and year, he says "keep A E" to only keep those columns. This un-highlights the values in columns B and D. Alternatively, he could have said "remove B D" to achieve the same goal.

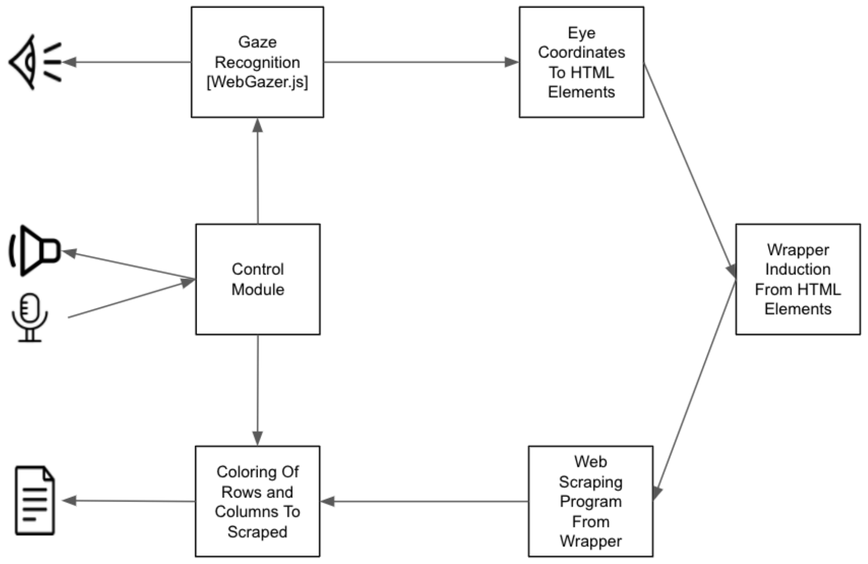
## Downloading Data

With the desired data highlighted and only the required columns kept, Ben finishes up by saying "download" to download the publication data to his computer as a JSON file. MWS enabled him to do this right in the context of Google Scholar without having to write any code!

## IMPLEMENTATION

### Overview

MWS is a Chrome browser extension written in Javascript. This enables it to run right in the browser on the website that a user wants to scrape data from.  Below is a diagram showing its system architecture:


Gaze Recognition [WebGazer.js]
Eye Coordinates To HTML Elements
Control Module
Wrapper Induction From HTML Elements
Coloring Of Rows and Columns To Scraped
Web Scraping Program From Wrapper

In the next sections, I explain how each of the system components work and how they are implemented.

## Gaze Detection

MWS uses an open source eye tracking library called *WebGazer.js* for gaze detection. WebGazer.js' eye tracking model self calibrates by mapping a user's mouse interactions to where on the website they are looking. Since MWS's interaction model does not involve mouse interactions, it is not able to take advantage of this and hence the need for an explicit calibration process.

Once the gaze detection is initiated, WebGazer.js provides the screen coordinates of the red circle that represents the user's detected gaze location. These screen coordinates are used to determine the Document Object Model (DOM) element under the red circle via the native *elementFromPoint* API. The DOM element is then used as an input to the wrapper induction process that generates the web scraping program.

## Wrapper Induction

*Wrapper induction* is the process of synthesizing a web scraping query from examples. Concretely, given a DOM element at the location of the user's gaze (*v*), we must find a set of *row elements* that represent the rows of the containing data table. We could naively assume that *parent(v)* is the row containing *v*, but often *v* could be deeply nested inside its containing row; we must determine which ancestor of *v* is likely to be the row.

Intuitively, MWS solves this problem by assuming that all rows share some similar internal structure. In particular, it expects most rows to contain a value for the given DOM element. (If there were no missing data, we'd expect *all* rows to contain data for this column).

Formally: assume a function *select(el, s)* which runs a Cascading Style Sheet (CSS) selector that returns the set of elements matching *s* within *el*. MWS generates a set of plausible candidates *P*, consisting of pairs of a row element and a CSS selector:

$P = \{ (r, s) \mid r \text{ in } ancestors(v) \text{ and } select(r, s) = \{v\} \}$

For each candidate *(r, s)* in *P*, we compute a weight function *w*, which is based on the number of siblings of *r* that have "similar structure", defined by checking whether running *s* within the sibling also returns a unique element.

$w(r, s) = |\{ r' \mid r' \text{ in } siblings(r) \mid select(r', s) \mid = 1 \}|$

MWS then chooses the candidate with the highest weight. In case of ties, the candidate closer to *v* in the tree (i.e., lower in the tree) wins. Given a winning candidate *(r, s)*, the full set of row elements is $\{r\} \cup siblings(r)$. Once the row elements are determined, each of them is traversed to find all their leaf nodes which correspond to column values.

### Speech Recognition & Synthesis

MWS uses the *Web Speech API* for speech recognition and synthesis. The table below shows all of MWS's voice commands and their functions:

| **Command** | **Description** |
|---|---|
| help | Displays all available commands |
| calibrate | Initiates gaze detection calibration |
| stop | Stops gaze detection when desired data is highlighted |
| continue | Continues gaze detection if wrong data is highlighted |
| keep | Keeps the mentioned columns e.g "keep A D" |
| remove | Removes the mentioned columns e.g "remove B E" |
| reset columns | Resets actions of keep and remove commands |
| download | Downloads scraped data as JSON file |
| close | Closes MWS |

When a user issues a command, it is written to MWS's dialog below the webcam feed. MWS responds to voice commands by both uttering a message (via speech synthesis) and displaying it in the dialog box.

## PERFORMANCE

Overall, I would assert that MWS is a success as a prototype of web scraping via gaze detection and speech. In particular, I am not aware of any existing work or research that leverages gaze detection to generate examples for wrapper induction. This is likely because of the lack of precision provided by eye tracking solutions, especially those that can be run in the context of a website. In the two sections below, I discuss the limitations of the two key aspects of MWS that reduce its effectiveness as a tool for web scraping: gaze detection and wrapper induction.

### Gaze Detection

WebGazer.js provides decent out of the box performance for gaze detection but is not precise enough to target individual column values. This realization is what made me switch to identifying row elements, selecting all available column elements and adding a mechanism to filter column elements. The biggest problem is that gaze detection is not steady, which is an artifact of how the human eyes work: they are never truly stationary. This makes it hard to target individual column elements which cover a small area of a website. More specialized eye tracking hardware such as the Tobii Pro Fusion eye tracking camera appear to provide smoother eye tracking but it is unclear whether they can be used in the context of a website which is necessary for the goal of web scraping.

### Wrapper Induction

MWS's wrapper induction process is most effective on websites whose data is presented as a collection of similarly-structured HTML elements. Certain websites, however, have designs that make it difficult to scrape data:

*Heterogeneous row elements* - Some websites break their content into rows, but the rows do not have a consistent layout, and contain different types of child elements. For example, the page design of HackerNews alternates between rows containing a title and rows containing supplementary data (e.g. number of likes and the time of posting). Because MWS only chooses a single row selector, when extracting by demonstration, it will only select one of the types of rows, and elements in the other types of rows will not be extracted.

*Large number of DOM elements* - For websites that render a very large number of DOM elements, the speed of the data highlighting significantly decreases. This is because the wrapper induction process queries the DOM which takes longer as the size of the DOM increases.

## USER STUDY

I have tested MWS with 5 people over the course of its development: 1 at the prototype stage, 1 at the implementation stage and 3 at the final stage. All participants had a background in Computer Science except 1. Furthermore, all participants had little to no experience with web scraping. The following sections discuss the user study I conducted with the 3 participants at the final stage. My goal was to assess its performance and discover barriers to its usability.

### User Tasks

The user study was conducted with each of the 3 participants individually over a recorded video conference. Each participant performed the tasks shown in the System Description section on Professor Randall Davis' Google Scholar profile:

1. Highlight publication data
2. Filter the highlighted data to only keep publication titles and publication years
3. Download the highlighted data

The participants were given a brief description about MWS but were not given a tutorial of how to use it beforehand. This was done to determine the effectiveness of MWS' interactive speech and text dialog feedback.

### Results

All the participants, who had little to no experience with web scraping, were able to use MWS to scrape data from a website which is very promising! I have divided the results I got into two groups:

1. *Shallow UI Issues*: this consisted of issues with MWS that can quickly be fixed in the next version. The most common one was that important voice commands, like “stop”, “continue” and “keep”, were only available upon issuing the “help” command. This is understandable as it was the first time they had seen and used MWS. All participants confirmed that subsequent uses which involve fewer issues with commands. Furthermore, all participants had trouble connecting the highlighted values to the column names needed for filtering. The current UI prepends column names, uppercase alphabet letters, to each column value but participants did not immediately notice them.

2. *Fundamental Barriers*: this consisted of issues that will require more time and effort to resolve. The most common one was the fact that careful timing is required to stop the gaze detection when the desired data is highlighted. Most participants had to stop and start the gaze detection, which highlights the data table corresponding to the DOM element at the red dot, several times before only publication data was highlighted.

## DEVELOPMENT

The development of MWS happened over 3 main stages which I describe in the section below.

### Design Stage

The initial design of MWS was focused on solely using gaze detection for web scraping. The vision was for uses to look at individual column values which would be fed into a wrapper induction algorithm that would generate a query for all available column values.

### Prototype Stage

While developing the prototype, I discovered that WebGazer.js’ precision was not sufficient to target individual column elements. Because the wrapper induction process identifies row elements in the process of generalizing column values, I was able to modify my approach and switch scraping row elements. With the row elements, I was able to traverse them to find their leaf nodes which corresponded to all available column values.

### Implementation Stage

To still enable scraping of only the desired column values, I tagged the column values in each row with column names (single, uppercase alphabet letter) and added voice commands with a two phrase vocabulary for filtering: “keep <column> <column>...” and “remove <column> <column>...”. This allowed users to utter commands like “keep A E” to only keep columns A and E highlighted  and “remove A E” to keep all but columns A and E highlighted.

### Final Stage

From the user test conducted at the implementat stage, I discovered that MWS was not usable without explicit instructions. To remedy this, I exposed the webcam feed created by WebGazer.js that shows whether a user is correctly positioned for gaze detection. More importantly, I added a command feedback mechanism that utilizes speech synthesis and text to respond to user commands and provide information about commands and what to do at each step of the process.

## LIBRARIES & API

The only external library that I used was WebGazer.js as the speech recognition and synthesis was available through the Web Speech API.

| **Library** | **Where is it available?** | **What does it do?** | **How well did it work?** | **Did it work out of the box? If not, what did you have to do to use it?** |
|---|---|---|---|---|
| WebGazer.js | webgazer.cs.brown.edu | Webcam eye tracking | It has decent performance, especially if its use involves mouse interactions which it uses to self calibrate | Most of the functionality I required worked out of the box. I had to modify the source code to move the webcam feed, which shows how a user's gaze is being detected, from its default top right position to the MWS interface. Because MWS does not use mouse interactions (which are used for self calibration), I had to create an in-website calibration mechanism to simulate mouse interactions. |
| underscore.js | underscorejs.org | Useful functional programming helper methods | Its popular library what works great | Worked out of the box (needed to implement debouncing during voice recognition) |