UNIVERSITY OF BIRMINGHAM

MSc Computer Science

Final Project

Improving Accessibility: Telephone-based Automatic Speech Recognition for Users with Disfluent Speech

Benjamin Hewett
886727
Supervisor: Dr Mark Lee
Git Repository:
https://github.com/BenHewett/MScProject

Acknowledgements

I'd like to thank my supervisor, Dr Mark Lee, for all of his support, advice and flexibility throughout the project. I would also like to thank Dr Noam Zeilberger for his useful advice following his inspection of my project and Mia Wright for her welfare support. I would also like to thank my family for their unwavering support throughout the whole of the master's programme, as well as friends Hollie and Ed for the occasional kick up the behind and technical advice respectively.

Table of Contents

Introduction

Automatic Speech Recognition (ASR) is an increasingly prevalent technology and is set to become more ubiquitous in the future given the recent development trajectory of speech-activated and/or controlled 'assistants'/devices, such as Siri, Alexa and Cortana, and wearable technologies, such as Apple's Watch. A recent article on the technology-focussed website Gizmodo highlighted that 40% of people currently conduct a voice search every day and that 30% of searches will be done without a screen by 2020 (Gizmodo, 2018).

However, with ubiquity comes the need for such technologies to be widely accessible to as many users as possible. Sadly, this accessibility still has some way to go before it reaches acceptable levels. In an article on science-focussed site Scientific American, Emily Mullin discusses the problems that many people with speech and voice disabilities face when trying to use speech or voice-controlled devices. She talks about people with Dysarthria (slow or slurred speech), people with swallowing problems and people who stammer/stutter (disfluent speech) (Scientific American, 2016). These problems arise because ASR systems are primarily designed for the process of fluent speech, and as such, have resulted in those with disrupted or disfluent speech being at a disadvantage when needing to use systems where the primary input method is speech. Mullin, rather pessimistically, concludes that "…as this technology…becomes more embedded in our daily lives…multitudes of people with speech and vocal problems will be excluded from connected 'smart' homes with voice-activated security systems, light switches and thermostats, and they might not be able to use driverless cars." (Scientific American, 2016). In essence, the increasing ubiquity of the voice to interact with the increasing amount of technology in our lives is leaving behind a segment of society.

This paper will chart a research project which identifies a set of accessibility problems generated by ASR systems when processing disfluent speech and/or are used by disfluent users; specifically, it will focus upon those users who stutter/stammer. It will initially elaborate upon the problem domain in order to settle upon a suitable focus before moving on to the iterative design, implementation and testing process behind the creation of a series of prototypes designed to investigate potential solutions. It will conclude with a 'final' system proposal which proports to solve the identified problems before providing an evaluation as to what worked and what didn't throughout the duration of the project.

Defining the Problem

As a person who stammers myself, I had some pre-conceived ideas about the scope of the project and which specific problems I wanted to address. Rather than focus my attention on the recognition of speech (the ASR software itself) I wanted to look at what happens *before* and *after* the recognition has taken place as I felt that these were areas which had not received enough attention and posed significant problems for the disfluent speaker.

My initial intuitions were as follows:

- The way in which ASR systems were designed to take input was highly problematic for the disfluent user. These problems often result in these systems becoming unusable for these people. I term this the *'input'* problem.
- That the input problem contributed to what I term the *'output'* problem: that of mis-recognitions of disfluent input.

However, I was also very aware that one person's challenges within a given domain are not necessarily shared by others, even those who experience the same condition. With this in mind, I set out to conduct some preliminary background research into the precise problem(s) that people who stammer face when using ASR systems, and also into the general attitudes people who stammer have towards ASR compared to those who don't; this research would also help to narrow down the scope of the project by highlighting the most common usages of ASR by the fluent and disfluent population.

With these initial considerations in mind, in this section I will do the following:

- Briefly outline distinctions between fluent and disfluent speech and describe, briefly, what ASR is and some use cases.
- Place my project within the landscape of current research on ASR and disfluency.
- Present the results and analysis of a survey I conducted into general attitudes towards ASR.
- Discuss the design, implementation, deployment and user testing of an unmodified ASR system which mimics the characteristics of typical commercial systems.
- Discuss the design, implementation and deployment of a data collection web application and the findings from the collected data.
- Offer a set of User Profiles which crystallise specific issues generated from the research conducted.
- Conclude with a set of functional and non-functional requirements which define the scope and goals of the project.

Speech Production

Human speech production is a complex process which requires the coordination of over 100 muscles in the chest and head. For the purpose of this paper, speech production will be usefully divided into two categories: speech which is produced *fluently* or *disfluently*.

Esmaili et al (2017), citing Starkweather (1997), usefully define fluent speech as being "…characterized by smoothness (lack of interruptions), normal speaking rate (not too fast or too slow), prosody (emotional intonation), and minimum mental effort (effortless speech)". They describe disfluency as "…any disorder in the fluency parameters [described above]". Stammering (or its synonym 'stuttering'), the prototypical form of disfluent speech, is characterised by the NHS as including the repetition of sounds or syllables – e.g. "mu-mu-mu-mummy", the prolongation of sounds – e.g. "mmmmmmmmummy" and/or the partial or complete blockage of sound where an utterance is 'blocked' and can't be produced. Furthermore, stammering "…varies in severity from person to person, and from situation to situation. Someone might have periods of stammering followed by times when they speak relatively fluently" (https://www.nhs.uk/conditions/stammering/).

Inspired by these definitions, I shall loosely define fluent speech production as speech which has a consistent *flow* with which sounds, syllables, words phrases (etc.) are joined together to form utterances; crucially, fluent speech can be said to be completely within the control of the speaker (i.e. requires little volitional effort). Contrary to this definition, I shall loosely define disfluent speech production as speech characterized by *repetitions* and *prolongations* of sound as well as *blockages to the flow of speech*; unlike fluent speech production, fluent speech production is *not* within the control of the speaker.

- Fluent speech production: speech characterized by a consistent flow.
- Disfluent speech production: speech characterized by an inconsistent flow; one defined by sound repetitions, prolongations and complete blockages.

Automatic Speech Recognition (ASR)

Automatic speech recognition is the process of taking human speech as an input, captured most commonly using a microphone, and processing – 'recognising' – the input into a string of text for use within a computing environment; the input is commonly used to then provide an output, the nature of which will depend upon the use case. Common usages of ASR are speech-to-text convertors, voice-activated assistants ('Siri', 'Alexa') and telephone-based customer service systems.

ASR/Stammering Survey

To further define the problem domain, I conducted a survey of general attitudes towards ASR within the fluent and disfluent population, as well as most common usages.

- Total participants: 138
    - Disfluent participants: 65 respondents
    - Fluent participants: 73 respondents

- Effectiveness of ASR
    - Fluent respondents: 60.5% found ASR effective or as effective as other systems
    - Disfluent respondents: 43% found ASR effective or as effective as other systems.

- Comfort with an increased prevalence of ASR in the future (0 = Very Uncomfortable, 10 = Very Comfortable)
    - Fluent respondents: average = 4.7
    - Disfluent respondents: average = 3.9

- Most common usages of ASR for fluent population:
    - When on the telephone (e.g. a bank): 74.60% (47 respondents)
    - To interact with my phone (e.g. Siri, Google): 66.67% (42 respondents)
    - To interact with my smart speaker (e.g. Alexa): 26.98% (17 respondents)

- Most common usages of ASR for disfluent population:
    - When on the telephone (e.g. a bank): 64.71% (33 respondents)
    - To interact with my phone (e.g. Siri, Google): 56.86% (29 respondents)
    - Using speech-to-text software: 25.49% (13 respondents)

Analysis:

The survey was taken by over one hundred participants, with a fair distribution of fluent and disfluent participants. Respondents were recruited mostly via personal contact and/or social media channels (Facebook, Twitter etc.), especially a Facebook-based support group for those who stammer run by the British Stammering Association. No personal data was collected, and the aim of the survey was simply to get a general collection of data regarding the usage of ASR in the two populations.

The survey produced some relevant results for my project. There was a significant difference between the perceived effectiveness of ASR in the fluent versus the disfluent population, with the disfluent population overall feeling that it was less effective for them. Furthermore, while not as significant a difference, the proposition that ASR would become more prevalent in the future was met with a slightly more sceptical response from the disfluent population than the fluent one. Overall, whilst the survey was quite narrow in focus, it does show a generally less favourable attitude towards ASR in the disfluent population compared to the fluent population.

Of further interest to my project was the most common usages of ASR in both population groups, with ‘on the telephone’ being the most common usage across the set of both

populations. Given this commonality, I decided to narrow the scope of my project to telephone-based ASR systems.

N.B: The full text of the survey, as well as the results, will be included in the Git repository for this project.

Alternative Systems

Given the decision to narrow the scope of the project to telephone-based ASR systems, it will be useful to briefly consider what alternative systems are available for those with speech impairments when using such telephone-based systems.

A very basic alternative system is sometimes offered which allows the user to use a special 'text phone', or the handset of their own telephone, in order to 'type' or key in what they want to say. Such systems are hampered by the lack of specialised hardware or the awkwardness of typing out a message on a keypad where each number represents several letters.

A more advanced system, which is more common (e.g. Argos, HMRC), is the Next Generation Text Service (NGT). NGT is aimed at deaf and speech-impaired individuals and assists them in communicating using the telephone. NGT consists of a network of three people: the speaker, an NGT human operator, and the recipient. It works as follows: using a Minicom, Uniphone or mobile phone application the speaker types what they want to say into their device; this is transmitted to the NGT human operator who reads the message and verbalises it to the recipient; the recipient verbally responds and the NGT operator types the response and sends it to the speaker. This process continues until the conversation is complete. Full details can be found at https://www.ngts.org.uk.

Whilst the NGT system is a vast improvement over the basic 'text phone' system, it is primarily designed for deaf individuals and those individuals for which speech is significantly impacted to the extent that verbal communication is ineffective. For people who stammer – many of which can be fluent for periods of time and even when disfluent, communication is still often achievable - it remains a convoluted and inadequate system.

As replacements for using primary ASR systems, these alternatives are unappealing.

In addition, the following legal information is taken from the Stammering Law website (http://www.stammeringlaw.org.uk/business/voicerecog.htm), a website created and maintained by a long-term British Stammering Association (www.stammering.org) volunteer and web master:

A stammer (or disfluency disorder) can be, and often is, interpreted as a disability under the Equality Act 2010. With this a given, there is a duty on the organisation to provide alternative means of interaction when the telephone is involved. Often these alternative means take the form of being offered the chance to speak to a human operative. It is

interesting to note that offering text access – as the NGT system does – is not enough and does not meet the duty placed upon the organisation to offer alternative arrangements.

As can be seen, the alternative arrangements on offer for systems which make use of ASR are generally aimed at those users for who verbal communication is ineffective or impossible. Disfluent users do not fall into this category and this explains why alternative systems are often found to be inadequate.

I will now consider where my project, as defined so far, fits within the current research.

Current Literature

A significant amount of research has been conducted into the use of ASR systems in a clinical environment with respect to disfluencies disorders. Rather than discuss every study, I will identify two strands of research and briefly discuss one example from each.

One strand of this research includes many studies that have been conducted into systems and technologies which assist the speech clinician with the task of *diagnosing* disfluency disorders. For example, in 'Automatic recognition of children's read speech for stuttering application' (Alharbi et al, 2017) the authors discuss a system which uses ASR to automatically recognise stuttering events so that the clinician can more accurately diagnose the severity of the condition in any given patient.

Another strand of research has focussed upon technologies which assist the speech clinician in *treating* disfluency disorders. For example, in their paper **'**An Automatic Prolongation Detection Approach in Continuous Speech with Robustness Against Speaking Rate Variations' (Esmali et al, 2017), the authors discuss their study into a system which detects prolongations in speech. This system is able to detect prolongations – based partly upon the speaking rate of the individual – and is then able to be used in order to support the clinician in teaching the client techniques to slow their speech; the system can be used to show how different rates of speech can impact the occurrence of prolongations within the client's modified speech.

Currently, there is not an ASR system which is 100% accurate. Given these current flaws in the technology, all users have become accustomed to the likelihood that their experience of any ASR system they choose to use will include having to repeat themselves due to the software not recognising what they have said; in a sense, the *anticipation* of misrecognition is ever present. For people who stammer, the likelihood, and so anticipation, of misrecognitions is exponentially higher than similar fluent users.

Within the research literature surrounding stammering, Bloodstein (1975) put forward a hypothesis regarding the 'anticipatory struggle' which, roughly, suggests that people who stammer "…believe that speaking is difficult and this belief in some way interferes with the smooth running of the processes that underpin fluent speech (Brocklehurst et al, 2012). The central idea behind this hypothesis is that people who stammer come to anticipate the

experience of stammering and that this anticipation somehow influences the likelihood of whether an experience of stammering is had. For example, a person who stammers could anticipate that they will block on all words beginning with the letter 'b' and, to some high degree, this happens.

Given the 'anticipatory struggle' hypothesis, Brocklehurst et al looked into the possible effects of the anticipation of word misrecognition on the likelihood of stuttering. Their study investigates "…whether the experience of stuttering can result from the speaker's anticipation of his words being misrecognized" (Brocklehurst et al, 2012). More specifically, the study focussed upon "…whether the experience of stuttering can be precipitated on specific words by instilling, in the speaker, the anticipation that those words will not be recognised by the recipient." (Brocklehurst et al, 2012). Brocklehurst el al therefore elaborate somewhat on Bloodstein's hypothesis by defining the subject of the anticipation as the recipient of any given instance of communication rather than the individual themselves anticipating that they will stutter. The results of this study showed support for the 'anticipatory struggle' hypothesis, as modified by the authors, in that "…the anticipation of communication failure can precipitate stuttering…" and that the findings "…also suggest that…the anticipation of communication failure is influenced both by feedback from their immediately preceding utterance(s) as well as by longer-term factors…" (Brocklehurst et al, 2012). These findings reinforce the prima facie notion that anticipation of stammering or, equally, the anticipation of stammering which then causes communication failure, can make communication harder for the person who stammers.

This brief review of the literature places this project outside of the current research focussed upon the clinical environment and as being partially inspired by the Brocklehurst et al paper. The results of the survey I reviewed earlier in this report showed that the general attitudes towards ASR by disfluent users is lower than the fluent user partly because of the perception that ASR is less effective for the disfluent population compared to the fluent. This perception could potentially be based upon the anticipation of being mis-recognised which itself would potentially lead to further disfluency which in turn would lead to further anticipation of mis-recognition. My project can therefore be seen as an extension of the ideas presented in Brocklehurst et al, (2012) as it directly addresses the issue of accessibility and the anticipation of mis-recognition.

ASR, Stammering and Accessibility

ASR is often incorporated into systems which are designed to process fluent speech; that is, they are designed to take as input a consistent flow of speech where interruptions to the flow are interpreted as the completion of the utterance ready for recognition. With reference to disfluent speech, this raises significant problems:

- Interruptions or blockages to the flow of speech cause the system to become confused as to when speech input has ceased, and recognition should occur
- Repetitions and prolongations of sounds change the shape or form of the sound which increases the likelihood of mis-recognitions and errors

In order to further refine the problem domain, I began by designed and implementing a simple web application which took a user's speech as input, processed the input through the ASR and displayed the results to the screen. My initial intuition was that disfluent users often have a problem with the input stage of using ASR due to the fact that the 'pause threshold' – the maximum duration of silence allowed before the system decides that input has ceased - for most systems is very short (typically 2-4 seconds):

ASR Comparison - Pause Threshold

- Google Search: 1 Second
- Siri (iPhone): 2 Seconds
- Siri (Apple Watch): 3 Seconds
- Argos (telephone): 3 Seconds

As can be seen from this selection of popular uses of ASR, the pause threshold is very short. For example, Google Search has a one-second pause threshold which, in practise, means that almost any pause in speech is interpreted as the end of the input and the system returns the results. An alternative method of use is provided for Google Search – key entry – but for systems like Siri on the Apple Watch, no such alternative is available which results in this system being very difficult to use for disfluent users; any kind of interruption to the flow of speech results in the system exiting and the user having to start again. Assuming that the user continues to experience disfluent speech, this can result in the process entering into an endless loop where the user is never able to complete their utterance in order to successfully use the system because they are consistently having to restart when the system interprets their disfluency as the end of the input.

In order to test this hypothesis, I designed a simple web application which implemented ASR and provided the function to be able to manually set and adjust the pause threshold value. The application took the participants voice as input and output the result from the recognizer to the screen.

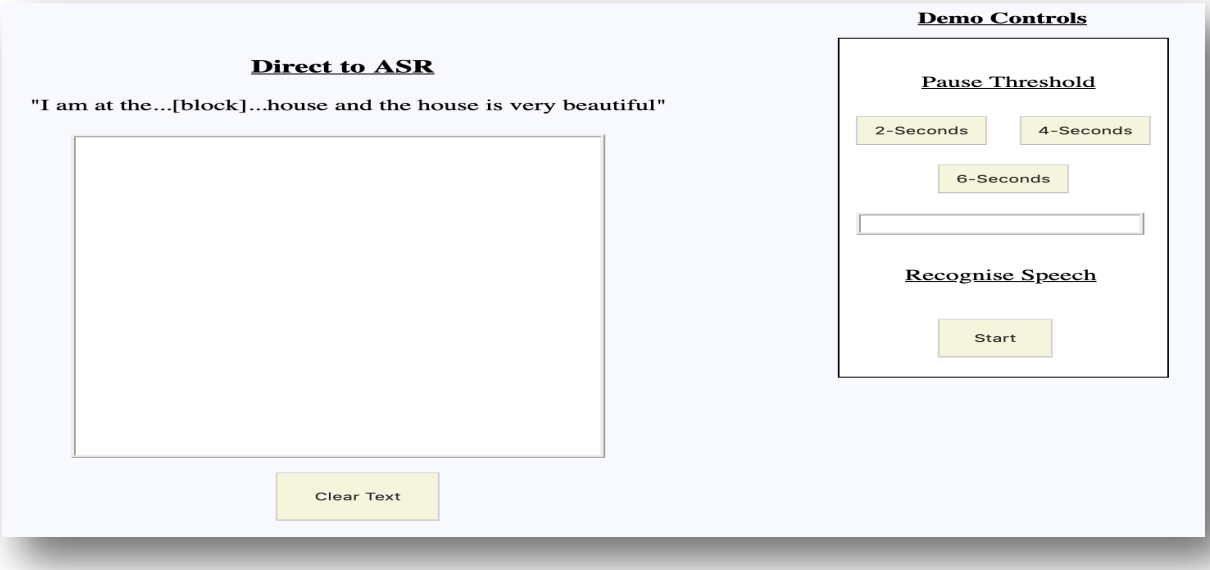


Figure 1 – Web application implementing ASR with manually adjustable pause threshold

Flow of Events

1. Select pause threshold
   [Threshold value is displayed beneath controls]
2. Press the 'start' button
3. Speak
   [System will output data once pause threshold is met]
   [Data displayed to the screen]
4. Complete process for all four passages

N.B: Full code description can be found in Appendix

I recruited five disfluent participants and, under my supervision, gathered user feedback. The results are displayed below:

PARTICIPANT	**FEEDBACK 1**	**FEEDBACK 2**
Irfan	"I found the 2-second pause almost impossible to use. Every time I stuttered it stopped and I had to start all over again."	"I find it really hard to use the speech option when searching with Google and so rarely use it simply because it never works for me. I don't see the point."
Ben	"The longer pause is a little easier because I don't feel as pressured into speaking fast but it still cuts out on me a lot before I've finished."	"As I often stammer when I'm starting to speak, it quite often cut out before I'd even said anything. This is precisely why I don't bother using ASR if I can help it."
Patrick	"I stammer quite openly so I didn't have any problem with it cutting out on me."	"I did have a problem with the fact that it wasn't very accurate when I stammered, it kept getting the words wrong."
Naheem	"I just couldn't be fluent enough to make it work consistently."	"The pressure I felt to be fluent was quite unpleasant and I'm sure made me stammer more!"
Norbert	"I rarely use these things [ASR] so I'm not surprised that it didn't work very well for me. The longer pause helped a little but I could still feel time pressure to hurry up."	"All I could think about when I blocked was that the time was ticking down and I had to say something before it decided I'd finished. Not nice!"

Figure 2 – Results of user testing on baseline ASR application

Analysis

As can be seen from the user testing process, all participants found the ASR difficult to use for three main reasons:

- The system interpreted interruptions in the flow of speech as being the end of the input process and returned the output
- The ASR mis-recognised disfluent passages of speech

- The short pause threshold increased the level of time pressure to uncomfortable levels

Conclusion

I began this section of the report with two intuitions:

- The way in which ASR systems were designed to take input was highly problematic for the disfluent user and that these problems often result in these systems becoming unusable for these people. I term this the *'input'* problem.
- That the input problem contributed to what I term the *'output'* problem: that of mis-recognitions of disfluent input.

I have demonstrated – via various exploratory means - that these intuitions have evidential force and that therefore they will be serve to narrow the focus of the project.

These problems will be demonstrated by the following user profiles and scenarios:

	Name: Joshua Smith Age: 15 Role: High School Student Speech Category: Disfluent Technological Comfort: Medium ASR Experience: Yes
ASR Usages	*Comments*
• Games Console • Siri on my iPhone	"It's really cool to turn my Xbox on with my voice. I just say, 'Xbox on' and it turns itself on. It seems to work really well for me, even when I stammer" "I find it really difficult to use Siri on my iPhone, especially when I'm around my friends. It doesn't like it when I stammer and says stupid stuff to me in reply. My friends laugh"

Figure 3 – User profile 1

	Name: Jane Anderson Age: 38 Role: Mother Speech Category: Disfluent Technological Comfort: Low ASR Experience: Yes
ASR Usages	*Comments*
• Alexa Speaker	"I have an Alexa in my house, and I tend to use it quite a lot when I'm at home with my young children. I'm not very good with technology but I like that there is one place where I can get music, radio and stuff from the internet." "The Alexa works ok for me, even when I stammer, when I'm only saying a few words but when I try to get some information from the internet, like the weather, and have to say a longer sentence, I often stammer and have to keep repeating myself."

Figure 4 – User profile 2

	Name: Dan Allen Age: 25 Status: Graduate Engineer Speech Category: Disfluent Technological Comfort: High ASR Experience: Yes
ASR Usages	*Comments*
• Games Console • Alexa Speaker • Siri on my iPhone • Google Search	"I find Siri on my iPhone almost impossible to use and I generally don't bother. I'd like to use it but it just can't understand what I say if I stammer. The same goes for Google." "I find the Alexa speaker pretty intuitive and easy to use and my Xbox is also pretty great at understanding me"

Figure 5 – User profile 3

Project Prerequisites

In order to design, develop and implement the required software, I would need to make some decisions about the technology I would use. The following are decisions I made:

- I would make use of the HTML and CSS mark-up languages in order to develop applications which will be web-based
- I would make use of the Python programming language and would do so for the following reasons:
    - Robust framework(s) for the development of web applications
    - Robust framework(s) for the implementation of ASR
    - Robust modules for data display and analysis

The decision to make use of the HTML and CSS mark-up languages is a fairly self-explanatory one: HTML and CSS are the de facto languages used for web development and any application which is to be made for web deployment must make use of these languages.

The decision to use Python rather than Java is not so self-explanatory and so I shall briefly describe my reasoning for doing so.

Given that my project would be primarily web-based, I required a robust set of tools to develop web-based applications. While Java provides web-based technologies (i.e. JavaServer Pages, Java Servlet API etc.) for building and deploying web applications, my decision to make use of Python was due to the simplicity of implementation of the web technologies and the variety on offer.

Python offers a wide range of full stack and non-full stack web frameworks, where a 'framework' is a group of technologies which usually include a HTTP application server, a storage mechanism such as a database, a template engine, a request dispatcher, an authentication module and an AJAX toolkit (Wiki.Python.org). Popular full stack examples include Django, TurboGears and Web2Py. Due to my project not requiring many of the technologies offered by a full stack framework – i.e. authentication, AJAX – and wishing to handle the database manipulation myself, I decided to go for a non-full stack framework. Popular examples of these are Bottle, CherryPy and Flask. All offer good solutions for my purposes but ultimately, I went for the Flask framework due to its simplicity and flexibility.

Python also offers a wide range of ASR technologies. As the focus of my project was not the ASR software itself, the choice of ASR technology came down to the simplicity of implementation. Python offers the following main modules for ASR implementation:

- Apiai
- AssemblyAI
- Google-Cloud-Speech
- Pocket Sphinx
- SpeechRecognition
- Watson-Developer-Cloud
- Wit

My final choice after testing each module (code to be included in project submission) by implementing each, was to make use of the SpeechRecognition module due to its simplicity and wide range of parameters available to tweak the ASR.

Overall, whilst the Python web technologies may be no better or worse than those offered by the Java language, the fact that Python also offers a wide range of ASR technologies as well as robust data display/analysis features made Python the sensible choice for my project.

On a personal note, my decision was partially informed by 1) the fact that my supervisor was a Python developer and would be able to advise me and 2) a desire to challenge myself by learning and making use of several new technologies, such as HTML, CSS and Python in order to expand my knowledge and skill set.

In order to develop the required applications in this project, I would need to learn the basics of the HTML, CSS and Python languages. In order to do this, I made use of text books from the *Head First* series: one for HTML & CSS and one for Python. These titles gave me a sufficient grounding in the respective languages in order for me to develop the required applications in this project.

Ultimately, the decision to use these technologies came down to the fact that these technologies were the best fit for the proposed project.

Process Model

Given the problem at hand, I decided upon using an iterative process model as this would help me to meet the requirements of the project.

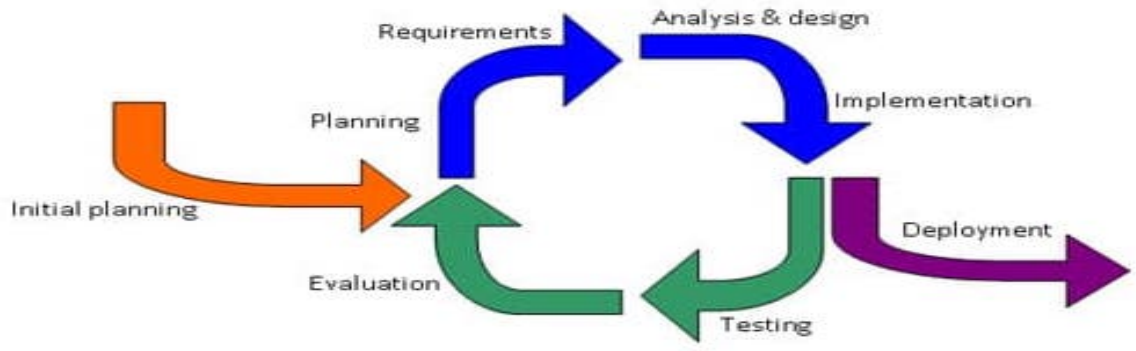


Figure 6 – the iterative process model

The majority of the project would focus upon the accessibility issues faced by disfluent individuals when using a telephone-based ASR system. Given the emphasis therefore placed upon user testing and feedback, an iterative model would best allow me to incorporate this user feedback into an on-going iterative process of design and implementation.

Other models considered include an iterative waterfall model, but it was decided that this was not suitable as the testing process is typically conducted before deployment and amendments to the requirements are less flexible using this model.

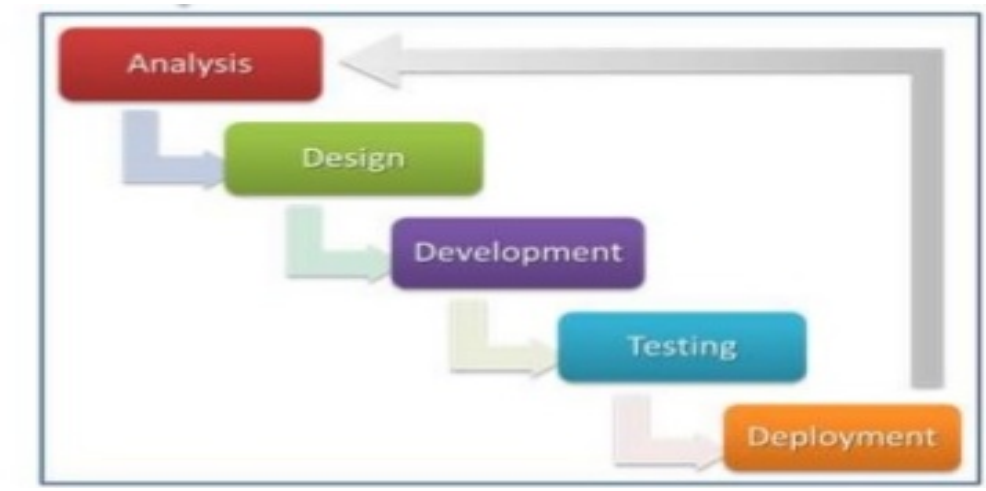


Figure 7 – an iterative waterfall process model

Data Collection Application

Given the project prerequisites, the first task of the project was to collect data samples. In order to collect the required samples of speech from participants, I needed to design, develop and implement an application which achieved this goal.

Data Collection Protocol

The data collection protocol defines the type of data I wish to collect and also the methodology which will be used to ensure that these goals are met:

- Phonetically-balanced passages of text

In order to determine the passages of text I would ask the participants to read, I contacted a speech and language therapist to see if there was a standard method for assessing the occurrence of disfluency; by using a similar method I could ensure that, given the inherent variability in the occurrence of stammering, I would get some recordings which included disfluency. In response I received information about 'standardised' reading passages which are often used to assess patients. These passages of text are 'phonetically balanced' so that as many of the speech articulators are used in as many combinations as is possible in order to ensure a thorough assessment of fluency. By making use of several of these passages, I could increase the possibility that I would get recordings which included disfluency. All passages are reproduced in full in Appendix C.

- Participant-led reading task

In order to ensure that I gave myself the best possible opportunity of attracting as many participants as possible, I decided to use a participant-led reading task in order to capture the audio recordings I required. This would involve the participant conducting a reading task at a time and place of their own choosing and the data being captured by the application I provided.

These considerations were defined as a set of functional and non-functional requirements.

System Requirements

Functional

1. **The system must be able to record speech provided by the participant**
   The main reason for the data collection application is to collect audio samples of participants reading selected passages of text. It must do this without error.
2. **The system should provide the opportunity for each participant to define their speech category, whether than be 'fluent' or 'disfluent'**
   In order to delineate between fluent and disfluent participants, each participant is required to disclose their speech category.

Each participant will be asked to read several phonetically-balanced passages of text.

Non-functional

1. **The system must be stable and robust**
   Data collection can be a challenge due to requiring participants' time and cooperation. Therefore, the application needs to be as robust and bug-free as possible before deployment in order to make the data collection as smooth and hassle-free as possible for the participant.

2. **The system must connect to a MySQL database and store relevant data**
   In order to store the audio provided by each participant, the system must connect to a database and store: filename, script ID, speech category, and WAV file parameters (frame rate, channels, sample width and audio data).

3. **The system should include clear and precise usage instructions**
   The system will be used by participants without supervision and therefore needs to be highly user-friendly and provide clear, step-by-step usage instructions.

4. **The system should provide the opportunity for each participant to record up to four distinct, phonetically-balanced passages**
   Each of these passages should be clearly delineated from other text on the page in order for it to be immediately clear which text the participant is asked to read.

5. **The system should be available as a web-based application using the Python Flask web-framework**
   Due to the geographical difficulties in attracting disfluent participants to take part in the data collection process, the system needs to be available online in order to reach as wide a participant pool as possible.

6. **The system must provide sufficient information as to gain the trust of each participant so that they are comfortable to proceed**
   Sufficient information must be provided in order to make each potential participant comfortable with taking part. Such information must include details about how each participants' data will be used and stored.

Implementation

Prototype

Figure 8 shows the information page, which explains the basics of the project and what is required of the participant, whereas figure 9 shows one instance of the data collection process.

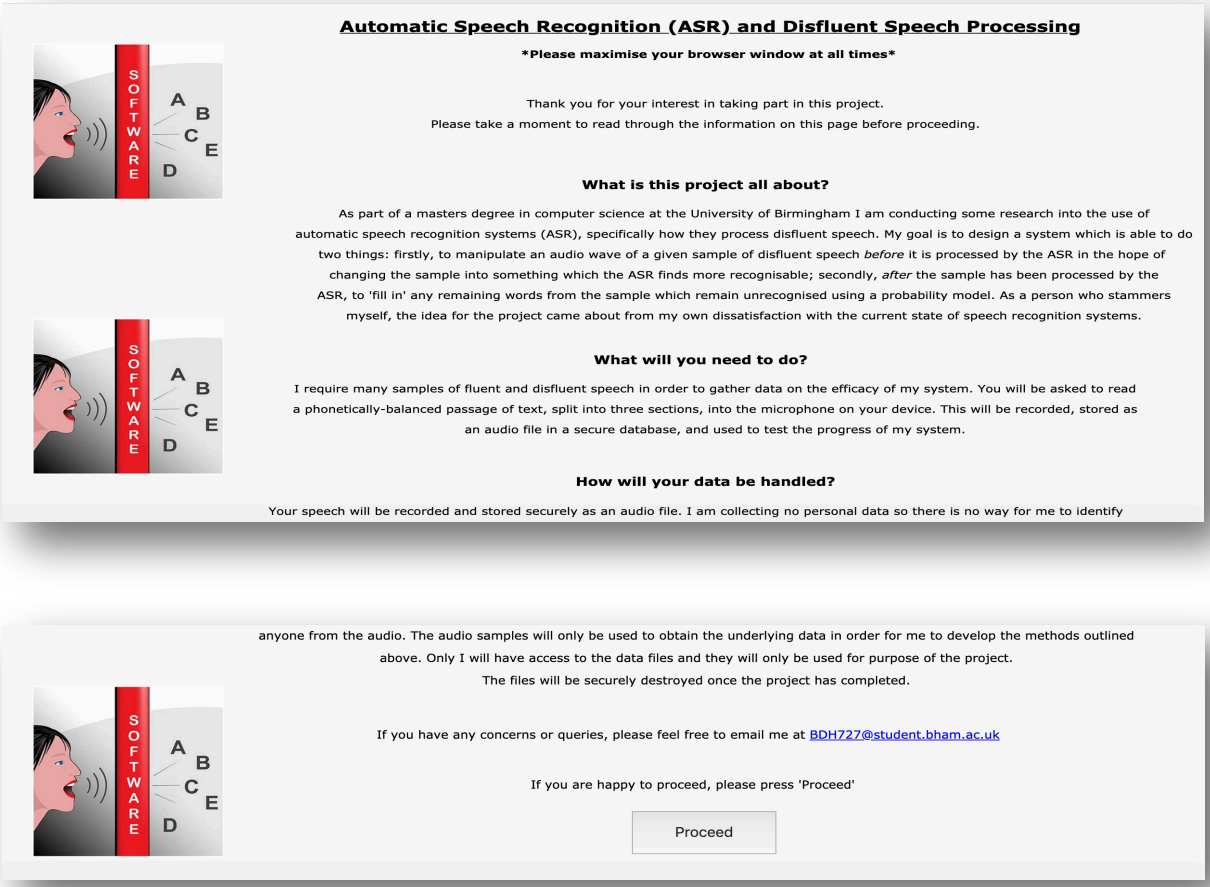


Figure 8 – 'Information' Page

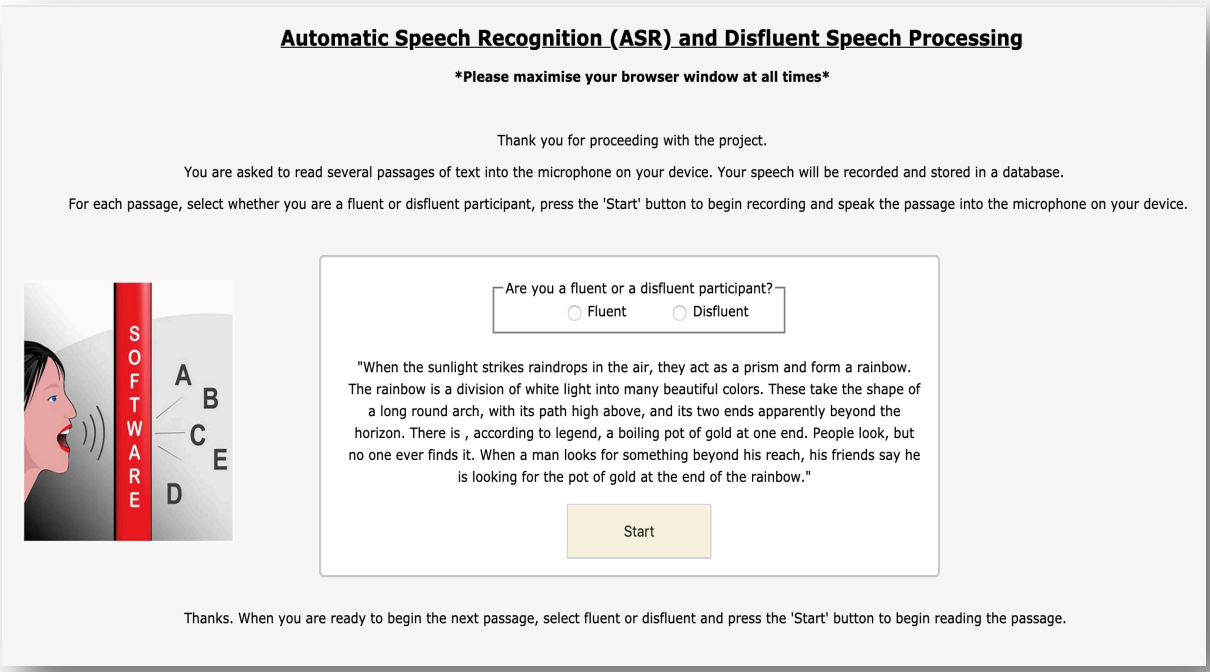


Figure 9 – Example of data collection

Flow of Events

1. Select speech type
2. Press the ‘start’ button to being recording (button label then becomes ‘Stop’)
3. Read the highlighted passage
4. Press the ‘stop’ button
   [Data stored in database]
5. Move onto next passage
   [Complete process for all four passages]

N.B: Full code description can be found in Appendix

Database

My database strategy involved the use of two tables. The first table stored the audio recordings and the second table stored the results of the data being processed through the ASR systems. Both tables were created within a MySQL database.

Table 1:

The ID field was designated as being the primary key in order to join with the second table in order to allow for queries which returned the audio data as well as it’s related ASR results. It was also set to auto increment in order to ensure unique entries for each audio recording.

The Frame Rate, Channels, Sample Width and Data fields are all elements of a WAV file and all are required in order to store a WAV file which can then be reconstructed upon being pulled from the database.

FIELD	**PARAMETERS**
ID	INT (11), UNIQUE, NOT NULL, PRIMARY KEY
Filename	VARCHAR (150), UNIQUE, NOT NULL
Script	VARCHAR (2), NOT NULL
Fluent	BOOLEAN, NOT NULL
Disfluent	BOOLEAN, NOT NULL
Frame Rate	INT (6), NOT NULL
Channels	INT (1), NOT NULL
Sample Width	INT (1), NOT NULL
Data	LONG BLOB, NOT NULL

Figure 10 – Audio Data Database Table

Table 2:

Similar to table 1, the ID field has been designated as the primary key, with auto-increment enabled, in order to be able to join the two tables in a useful way.

The ASR Script Word Count, ASR Script Text, ASR Accuracy Rate, Misrecognition Count and Misrecognitions fields all hold data which is produced by the processing of each audio file through the chosen ASR system.

FIELD	**PARAMETERS**
ID	INT (11), UNIQUE, NOT NULL, PRIMARY KEY
Filename	VARCHAR (150), UNIQUE, NOT NULL
Fluent	BOOLEAN, NOT NULL
Disfluent	BOOLEAN, NOT NULL
Script ID	VARCHAR (2), NOT NULL
Script Word Count	INT (6), NOT NULL
Script Text	TEXT
ASR Script Word Count	INT (6), NOT NULL
ASR Script Text	TEXT
ASR Accuracy Rate	FLOAT, NOT NULL
Mis-recognition Count	INT (6), NOT NULL
Mis-recognitions	VARCHAR (200), NOT NULL

Figure 11 – ASR Results Database Table

Application Deployment

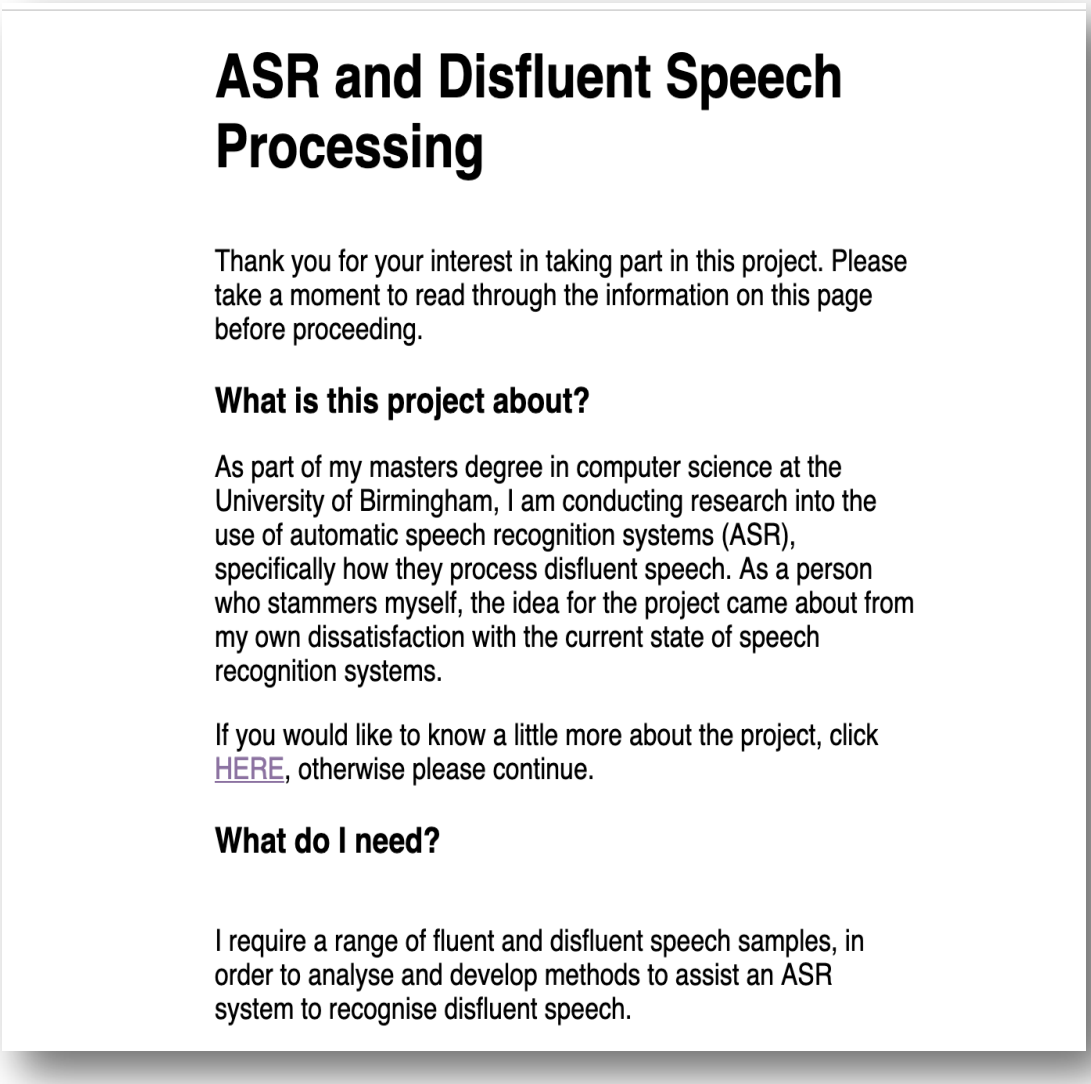
In order to deploy the data collection application, I would need to host it on a web hosting service. For this I decided to use PythonAnywhere due to the fact that it handles all of the server-side requirements.

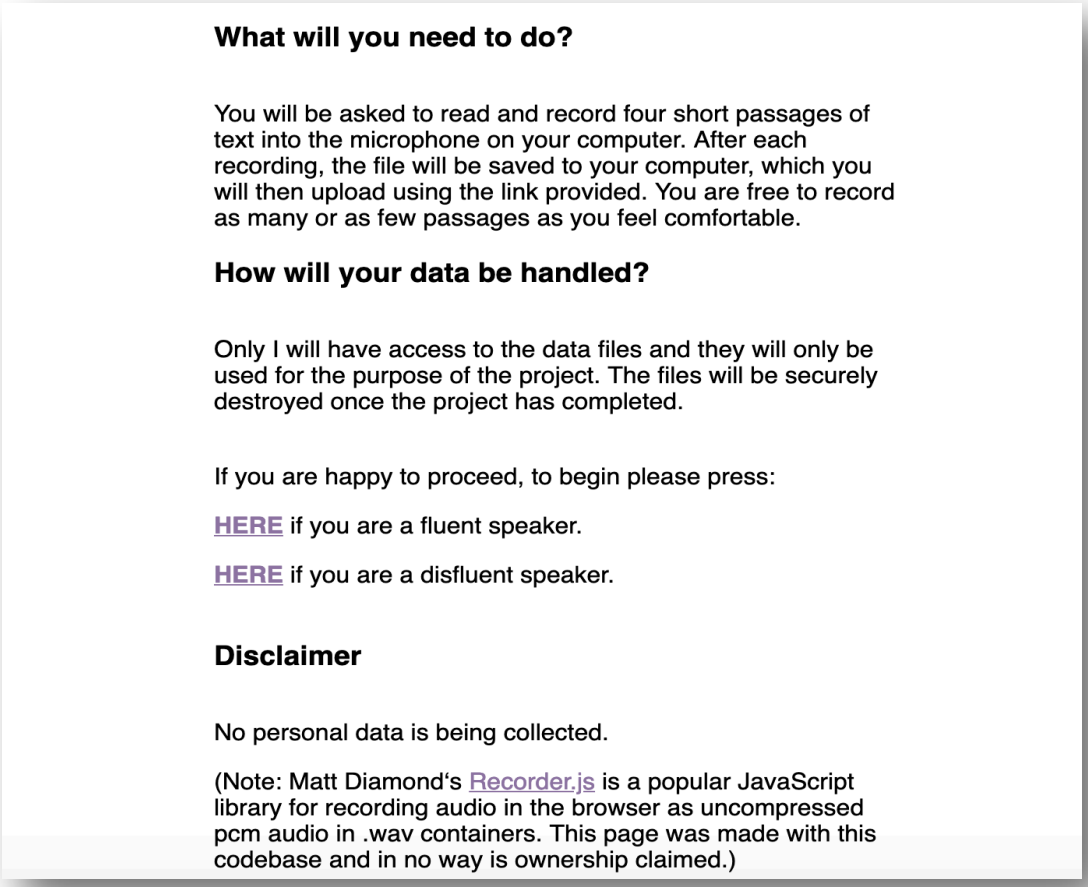
However, during the deployment process I ran into a problem with the application I had developed. The back-end Python script which I had written to record audio was based upon the PyAudio module and, sadly, this module was incompatible with PythonAnywhere. After some research, I discovered that for the script to be able to run and for me to record audio it would need to reside within the client application – in my case the users web browser – in order to be able to interact with the user's microphone. In light of this problem, I looked into several solutions, including packaging the application for distribution to each individual user, but ultimately decided to refactor my application to make use of JavaScript in order to solve the problem (JavaScript works client-side from within the web browser). I took this

decision because I felt that requesting each and every user to download, install and run the application would require too much knowledge, especially given that many potential participants had expressed a lack of technical knowhow. I felt that a web application which was 'ready to use' would be far more effective at attracting participants.

Given these considerations, and given the time constraints on the project, I decided to use an open-source JavaScript library in order to provide the audio recording functionality rather than taking the time to learn sufficient JavaScript myself. Therefore, all JavaScript included within my project is not authored by myself and I declare no ownership of it. Therefore, I refactored the HTML and CSS code from the prototype application so that it would work with the JavaScript audio recording module. The deployed application now meets the same requirements.

Figure 12 and 13 shows the redesigned application.





What will you need to do?

You will be asked to read and record four short passages of text into the microphone on your computer. After each recording, the file will be saved to your computer, which you will then upload using the link provided. You are free to record as many or as few passages as you feel comfortable.

How will your data be handled?

Only I will have access to the data files and they will only be used for the purpose of the project. The files will be securely destroyed once the project has completed.

If you are happy to proceed, to begin please press:

HERE if you are a fluent speaker.

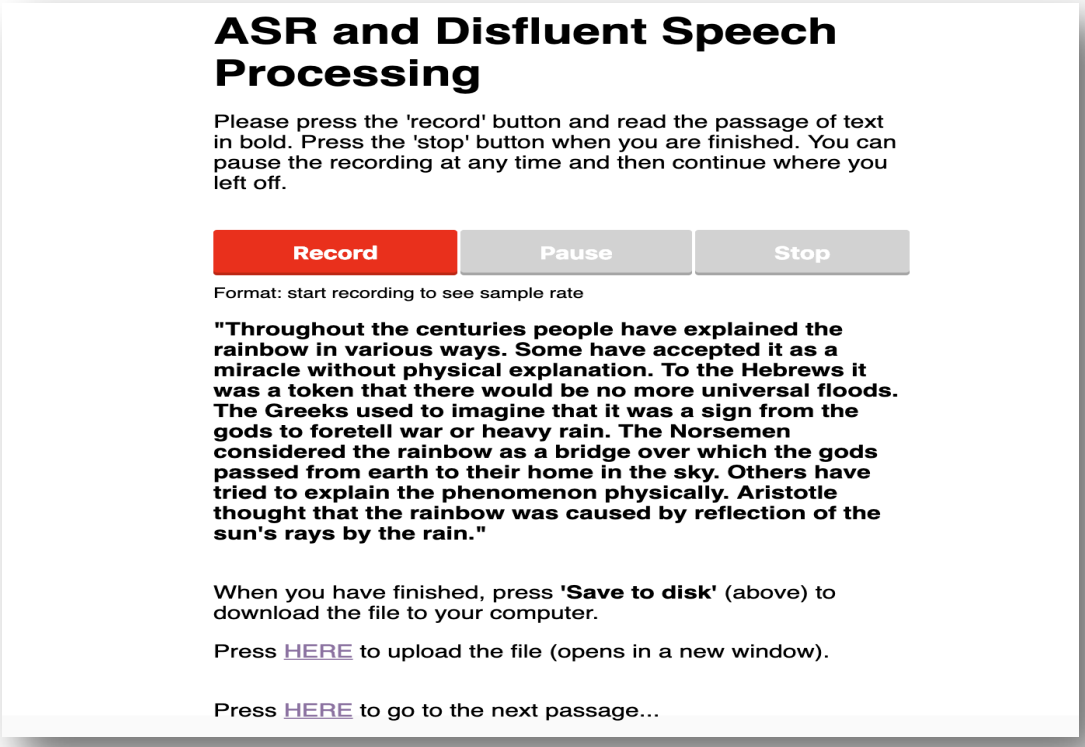
HERE if you are a disfluent speaker.

Disclaimer

No personal data is being collected.

(Note: Matt Diamond's Recorder.js is a popular JavaScript library for recording audio in the browser as uncompressed pcm audio in .wav containers. This page was made with this codebase and in no way is ownership claimed.)

Figure 12 – The Information page



ASR and Disfluent Speech Processing

Please press the 'record' button and read the passage of text in bold. Press the 'stop' button when you are finished. You can pause the recording at any time and then continue where you left off.

Record | Pause | Stop

Format: start recording to see sample rate

"Throughout the centuries people have explained the rainbow in various ways. Some have accepted it as a miracle without physical explanation. To the Hebrews it was a token that there would be no more universal floods. The Greeks used to imagine that it was a sign from the gods to foretell war or heavy rain. The Norsemen considered the rainbow as a bridge over which the gods passed from earth to their home in the sky. Others have tried to explain the phenomenon physically. Aristotle thought that the rainbow was caused by reflection of the sun's rays by the rain."

When you have finished, press **'Save to disk'** (above) to download the file to your computer.

Press HERE to upload the file (opens in a new window).

Press HERE to go to the next passage...

Figure 13 – An example of one of the data collection pages

Given this refactoring, I decided to forgo PythonAnywhere and deploy the application to Google's App Engine platform. The deployment process for GAE was straightforward – a case of uploading the HTML, CSS and JavaScript files, adding a 'yaml' configuration file to handle the dependencies, and launching the application.

Flow of Events

The flow of events for the data collection application can be seen in the following diagram:

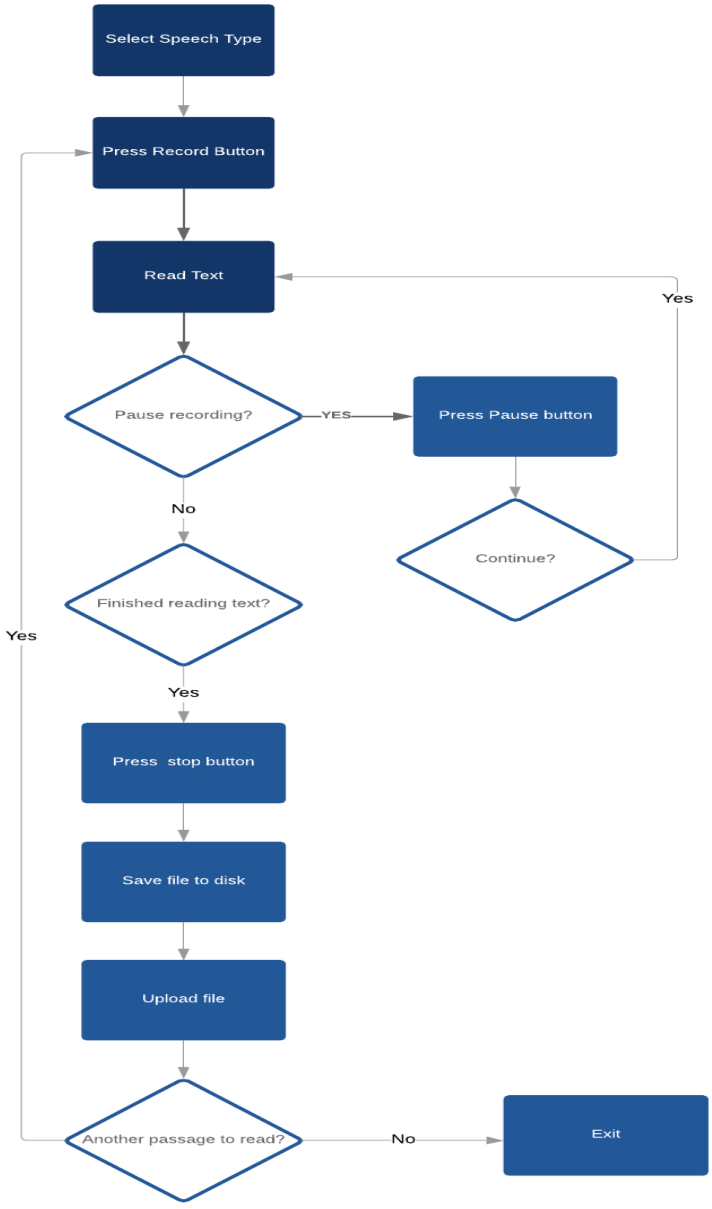


Figure 14 – Data Collection Application flow of events

Data Processing and Analysis

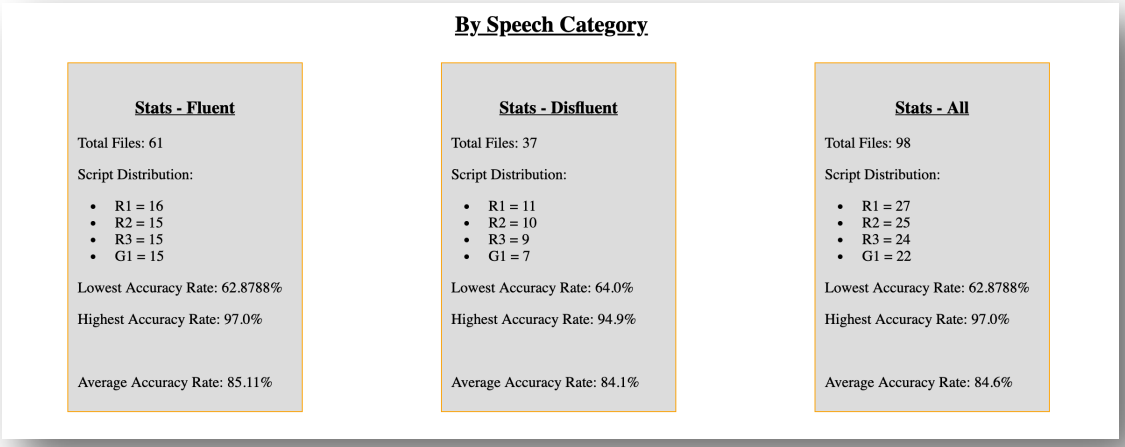
Once the data had been collected, it needed to be processed and analysed.

A total of 32 people agreed to provide data for my project, resulting in the collection of 125 audio files in WAV format. All data was stored securely in a MySQL database.

ASR Processing

All files were processed individually through the Speech Recognition module (https://pypi.org/project/SpeechRecognition/). Full descriptions of the code used to achieve this and calculate the statistics can be found in the appendix.

Full results can be seen below:



By Speech Category

Stats - Fluent

Total Files: 61

Script Distribution:

- R1 = 16
- R2 = 15
- R3 = 15
- G1 = 15

Lowest Accuracy Rate: 62.8788%

Highest Accuracy Rate: 97.0%

Average Accuracy Rate: 85.11%

Stats - Disfluent

Total Files: 37

Script Distribution:

- R1 = 11
- R2 = 10
- R3 = 9
- G1 = 7

Lowest Accuracy Rate: 64.0%

Highest Accuracy Rate: 94.9%

Average Accuracy Rate: 84.1%

Stats - All

Total Files: 98

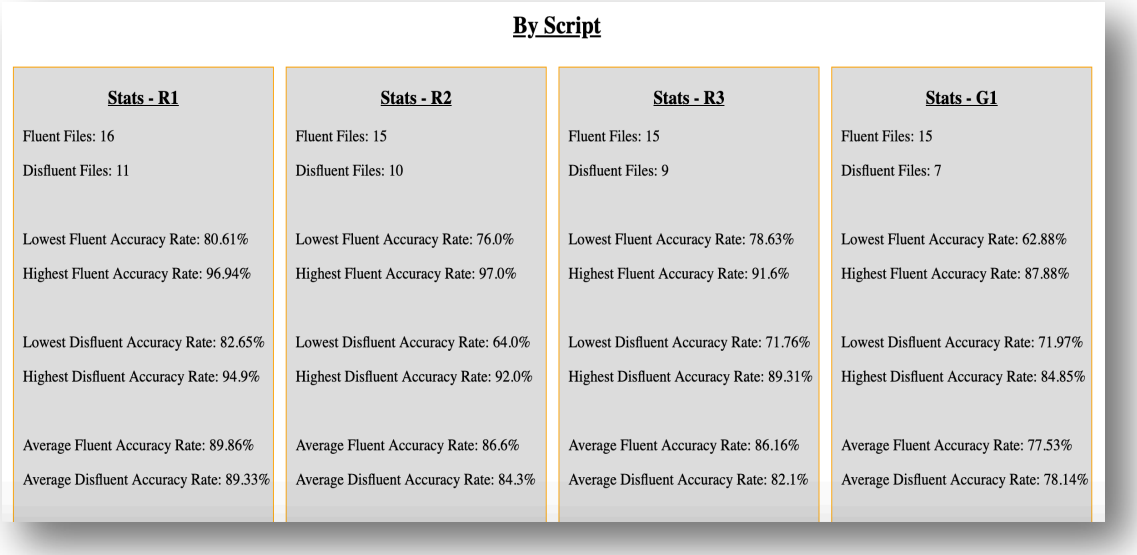
Script Distribution:

- R1 = 27
- R2 = 25
- R3 = 24
- G1 = 22

Lowest Accuracy Rate: 62.8788%

Highest Accuracy Rate: 97.0%

Average Accuracy Rate: 84.6%



By Script

Stats - R1

Fluent Files: 16

Disfluent Files: 11

Lowest Fluent Accuracy Rate: 80.61%

Highest Fluent Accuracy Rate: 96.94%

Lowest Disfluent Accuracy Rate: 82.65%

Highest Disfluent Accuracy Rate: 94.9%

Average Fluent Accuracy Rate: 89.86%

Average Disfluent Accuracy Rate: 89.33%

Stats - R2

Fluent Files: 15

Disfluent Files: 10

Lowest Fluent Accuracy Rate: 76.0%

Highest Fluent Accuracy Rate: 97.0%

Lowest Disfluent Accuracy Rate: 64.0%

Highest Disfluent Accuracy Rate: 92.0%

Average Fluent Accuracy Rate: 86.6%

Average Disfluent Accuracy Rate: 84.3%

Stats - R3

Fluent Files: 15

Disfluent Files: 9

Lowest Fluent Accuracy Rate: 78.63%

Highest Fluent Accuracy Rate: 91.6%

Lowest Disfluent Accuracy Rate: 71.76%

Highest Disfluent Accuracy Rate: 89.31%

Average Fluent Accuracy Rate: 86.16%

Average Disfluent Accuracy Rate: 82.1%

Stats - G1

Fluent Files: 15

Disfluent Files: 7

Lowest Fluent Accuracy Rate: 62.88%

Highest Fluent Accuracy Rate: 87.88%

Lowest Disfluent Accuracy Rate: 71.97%

Highest Disfluent Accuracy Rate: 84.85%

Average Fluent Accuracy Rate: 77.53%

Average Disfluent Accuracy Rate: 78.14%

Figure 15 and 16 – ASR results by speech category and ASR results by script ID

Analysis

As the results show, the statistics compared between fluent and disfluent users are not showing the kind of distinction I imagined I would see with the average accuracy rate only differing by 1% (85.1% vs 84.1%). My expectation was that I would see a significant difference between the average accuracy rate for disfluent users and fluent users, with the disfluent average being lower. Investigating this – discussing it with participants – I discovered that the reading task which I had designed allowed them to prepare and to ensure that their environment was free of additional stressors and that this meant that they were more or less fluent in that situation. This highlights a flaw in the design of the data collection protocol: I did not consider carefully enough the environment under which the data collection would take place, and this has resulted in the collected data being less useful than originally hoped. This is one area where improvement would be required.

However, even with this taken into account, the collected data still provided me with sufficient disfluent data to use in order to continue the project.

The 'Input' Problem

The 'input' problem, as defined by user testing and feedback, is the problem that the pause threshold for most ASR-based systems is very short and that this is a major accessibility barrier for disfluent users.

In order to propose a solution to this problem, I decided to use an iterative methodology in order to make use of the user testing process in order to improve my designs.

With this in mind, I design a simple web application which represented a telephone-based ASR system. For ease of use and for testing purposes, I built this application with a display area included in order to be able to view the output of the ASR system.

Prototype 1

The first prototype included the option to increase the pause threshold of the ASR. I decided to begin at 5 seconds and to offer three further options in increments of 5 seconds, culminating in a 20 second pause threshold.

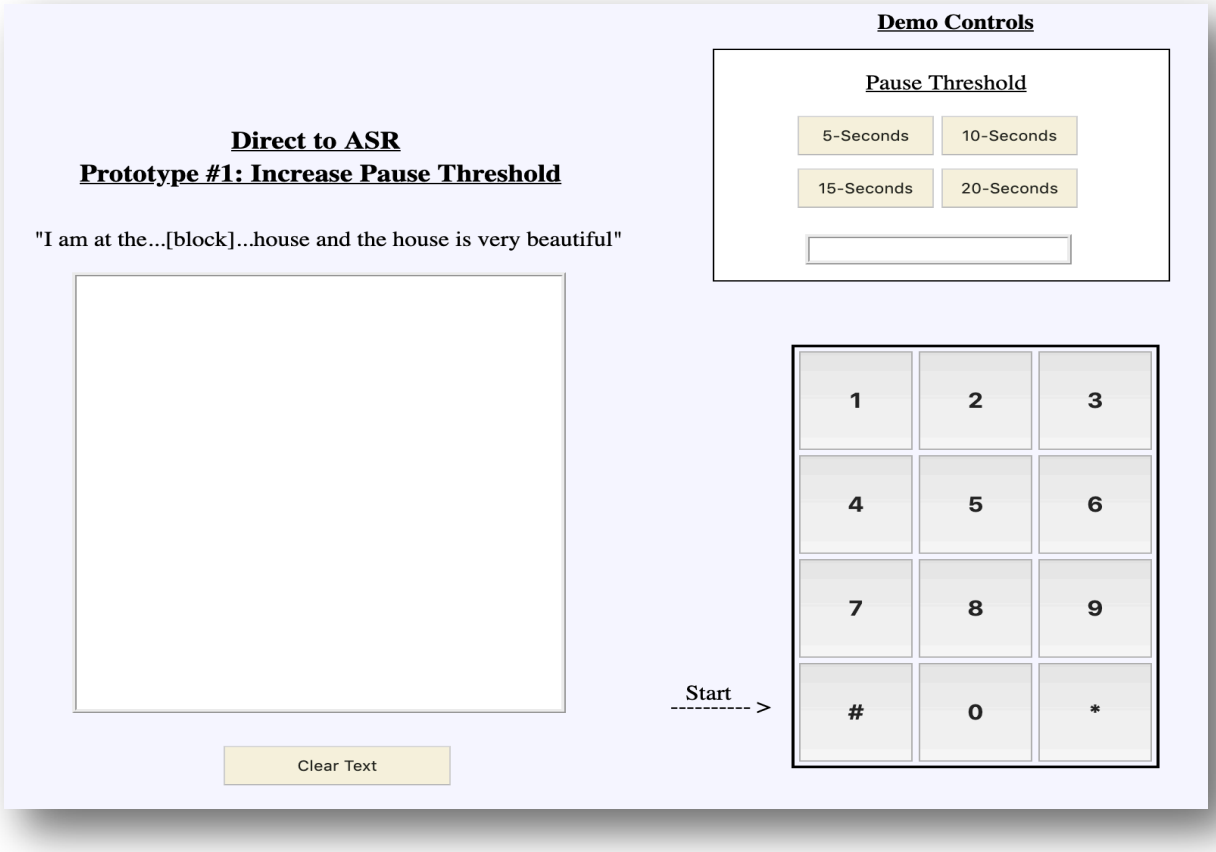


Figure 17 – Prototype 1, an increase to the pause threshold

Making use of the same five participants, and under my supervision, I asked each one to follow the sequence of events I provided (below) and to provide feedback about their experience of 1) the pause threshold and 2) the ease of use of the ASR system in general.

Flow of Events

1. Select the required pause threshold value
   [Pause threshold value displayed on screen]
2. Press the '#' key on the keypad to start the ASR recognizer
3. Read the required passage
   [ASR recognises input]
4. ASR result displayed in the output area
5. Press the 'clear text' button to clear the output area
   [Repeat process making use of all four pause threshold options]

PARTICIPANT	**PAUSE THRESHOLD**	**ASR**
Irfan	"Overall, having tried them all, I would say that the 10 second pause threshold was decent, and it only cut me off once when I blocked quite badly"	"The 20-second threshold was much, much too long and left me waiting. The system was easier to use but didn't feel satisfying due to still feeling pressure."
Ben	"I can block quite severely at times so only the 15-second threshold really took that into account."	"I found the system much easier to use! I didn't feel as much time pressure as before, knowing I had longer, although that pressure wasn't removed entirely."
Patrick	"As before, I stammer quite openly so the pause threshold wasn't really a big problem for me. In fact, a longer one would leave me waiting for it to finish."	"I had a real problem waiting for the ASR to finish listening and to show me the results."
Naheem	"A 5-second threshold was good for me. Anything longer and I'd be waiting around for ages."	"I'm quite covert with my stammer [user word swaps and hides his stammer if possible] so any longer than 5-seconds left me waiting for a long time for it to finish listening."
Norbert	"10 seconds seemed like a good middle ground for me. It still cuts out on me from time to time, but it certainly lessened the problem."	"I found the system easier to use with a longer pause but I still felt time pressure which likely increased the chance of me blocking."

Figure 18 – Prototype 1 user feedback

Analysis

- All users found the system easier to use with a longer pause threshold (5+ seconds)
- Users had various preferences for the pause threshold value:
  - One user preferred the 5 second value
  - Two users preferred the 10 second value
  - One user preferred the 15 second value
- The system setup still resulted in users being cut off mid-utterance and before they completed what they were saying which resulted in inaccurate outputs
- Time pressure was reduced for some users but not eliminated
- When the pause threshold was longer than 5 seconds, if the user was more fluent, the wait time for the output was deemed unacceptable

Considerations for Prototype 2

- The problem of the system cutting off user's mid-utterance remained
- A problem was introduced regarding the output waiting time being too significant given increased fluency
    - Computational efficiency will also be adversely affected by a longer pause threshold
- The felt time pressure of the users was reduced but not eliminated, which could increase disfluency in the user

Prototype 2

Taking into account the user feedback from prototype 1 – especially regarding the continued cut-offs and the unacceptably long wait times for output – the second prototype removed the pause threshold. All ASR systems have to have a pause threshold so that they know when to stop listening and to return a result. To get around this problem, I designed this prototype to record the user's speech as audio *and then* to process this audio through the ASR system.

Following the same procedure as before, using the same five participants under my supervision, I asked each one to follow the sequence of events I provided (below) and to provide feedback about their experience of 1) the control scheme and 2) the ease of use of the ASR system in general compared to the previous iteration.

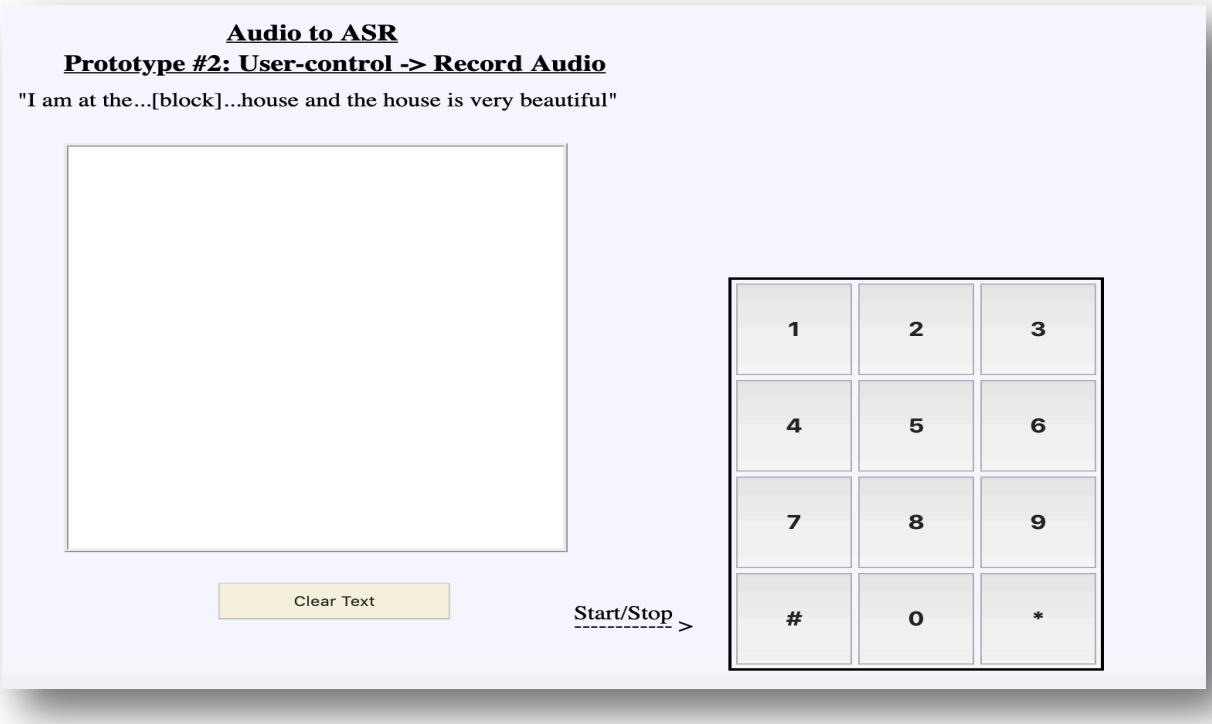


Figure 19 – Prototype 2, recording user speech before ASR processing

Flow of Events

1. Press the '#' key on the keypad to start the recording process
   [Microphone activated and recording begins]
2. Read the required passage/say what you want
3. Press the 'stop' button when finished speaking
   [Recording ceases and audio file automatically processed through ASR]
4. Output displayed on screen in the output area
5. Press the 'clear text' button to clear the output area

PARTICIPANT	**Audio Recording**	**ASR**
Irfan	"I really liked the fact that I had more control of the process and that the time pressure was removed because this does make me stammer more."	"Using the whole system was a lot easier than before because there wasn't any pressure on me to be fluent."
Ben	"I felt a tangible relief that the time pressure was no longer present."	"The accuracy of the result wasn't improved but it was a lot, lot easier to use it because I wasn't rushed."
Patrick	"This didn't really improve things for me as I stammer openly anyway and don't block much. It was good that the time pressure factor was completely removed though."	"Much easier to use and the wait time for the result wasn't as long as the last one when the pause threshold was too long."
Naheem	"This version definitely solved the problems that I found in the last one simply because there was no pause threshold."	"Much easier to use and allowed me to use without problem. Pity all speech recognition doesn't do this!"
Norbert	"This worked well. I forgot to press the stop button a couple of times so this might take some getting used to."	"Apart from the button presses - maybe having the start and stop buttons on different keys? - I found the system very simple to use."

Figure 20 – Prototype 2 user testing

Analysis

- The problem of cut-offs was removed with this prototype resulting in positive user feedback from all
- The output wait time was lessened due to the removal of the pause threshold
- The felt time pressure on the user was removed due to the removal of the pause threshold
- Output wait time still existed if audio file included a lot of silence resulting from interruptions to the flow of speech for significant lengths of time
- Output accuracy was unaffected by this modification

Considerations for Prototype 3

- Computational processing time could theoretically become unacceptable if audio recording included a lot of silence
- Output accuracy was unaffected as all disfluency was captured on the audio file

Prototype 3

Taking into account the user feedback from prototype 2 – especially regarding the unaffected output accuracy – the third prototype added in a further element of control in that it allowed the user to pause the recording process and then be able to restart the recording process from where they left off. Many people who stammer are aware of when they enter into a block and, with varying degrees of haste, are able to stop, reduce tension in the area of the block and to continue. With this in mind, the option to pause and restart the recording may off the user an opportunity to do this while the recording process is temporarily suspended. With this in mind, fewer disfluencies might be captured and processed by the ASR.

Following the same procedure as before, using the same five participants under my supervision, I asked each one to follow the sequence of events I provided (below) and to provide feedback about their experience of 1) the control scheme and 2) whether they felt able to pause the system when they blocked and 3) the ease of use of the ASR system in general compared to the previous iteration.

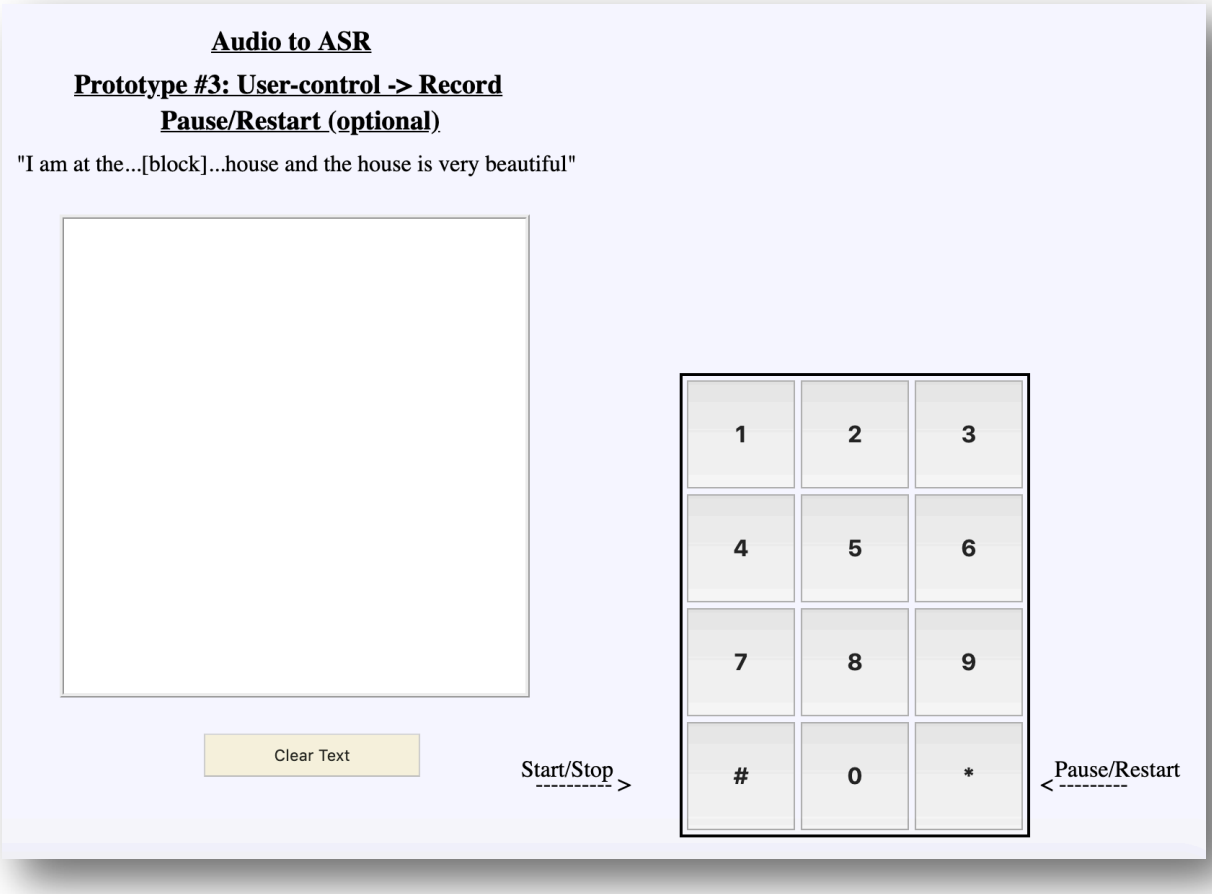


Figure 21 – Prototype 3, recording user speech with additional pause/restart mechanism

Flow of Events

1. Press the '#' key on the keypad to start the recording process
   [Microphone activated and recording begins]
2. Read the required passage/say what you want

If required:

3. Press the '*' button to pause the recording
4. Press the '*' again to restart the recording

Else:

5. Press the '#' button again when finished speaking
   [Recording ceases and audio file automatically processed through ASR]
6. Output displayed on screen in the output area
7. Press the 'clear text' button to clear the output area

The flow of events for this prototype can be seen in the following diagram:

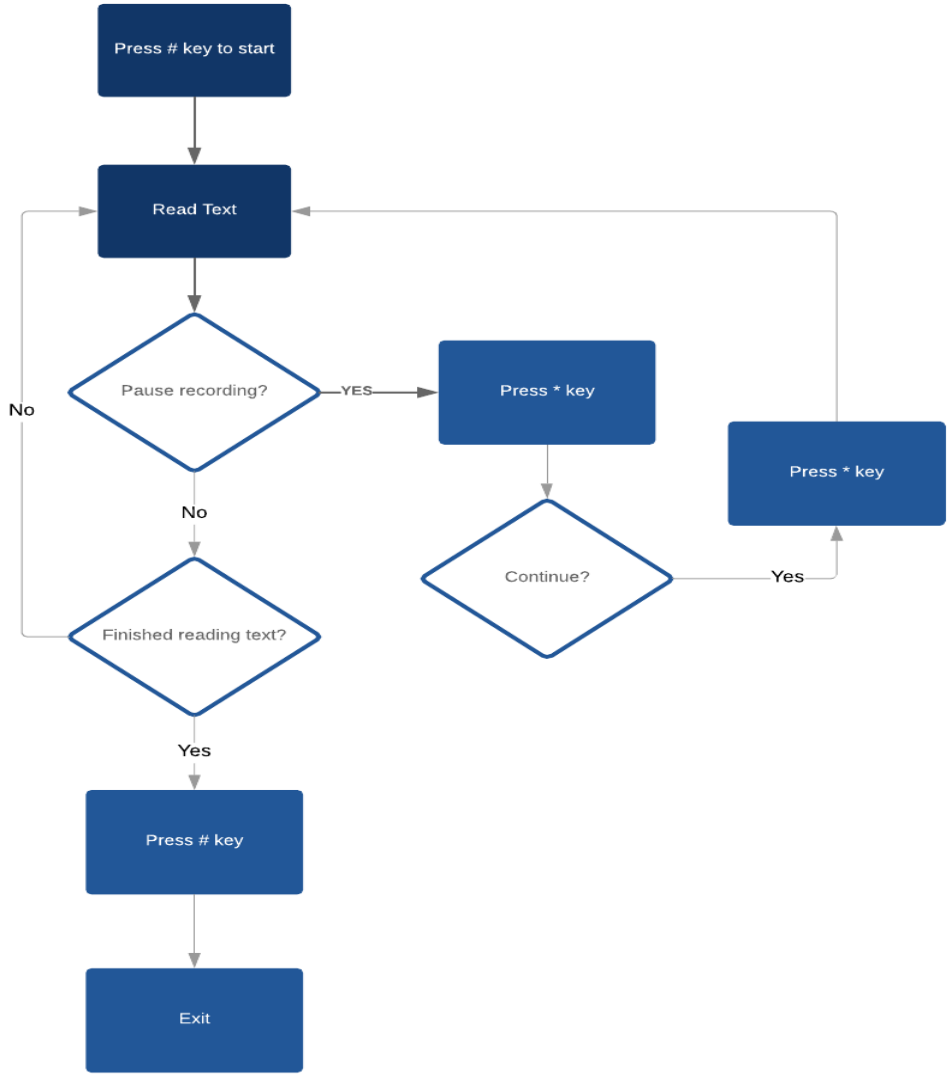


Figure 22 – Prototype 3 flow of events

PARTICIPANT	**PAUSE/RESTART FUNCTION**	**ASR**
Irfan	"It works as expected and I was able to, on a couple of occasions, pause when I stammered. It worked quite well actually."	"The system as a whole felt much more under my control rather than before when I felt I had to adjust to it in order for it to work well."
Ben	"I loved the feeling of control these controls gave me. I think having them on two buttons was a good choice too as four buttons would be very confusing."	"This really took away any sense of time pressure I felt with the other versions and I felt I could express myself fully and without constraint."
Patrick	"I didn't really use the pause option much as I just stammered away normally."	"In much the same way as the other versions were easier to use than the standard system, this was just as easy although for me it didn't offer much improvement over the last version."
Naheem	"I liked the option being available to pause the recording but I'm sceptical about how much I would use it. I definitely think it's a positive addition though."	"Overall, even though I'm sceptical about how much I would use the pause option, I think it's inclusion adds further to the sense of control and therefore the removal of pressure."
Norbert	"This worked really well! I'm surprised as I was sceptical but in use it's really easy to control and use."	"Much improved!"

Figure 23 – Prototype 3 user testing

Analysis

- Potential for long audio files reduced due to the additional control afforded to the user with the pause/reset function
- Sense of control/freedom increased with all users expressing positive feedback about the removal of time pressure and the opportunity to express themselves without restraint
- Positive feedback regarding the use of two buttons for the four functions – start, stop, pause, restart - rather than each function having its own button
- Potential for periods of silence to appear in the audio which will increase the processing time when put through the ASR system

Considerations for Prototype 4

- Prototype 3 seems to have hit a sweet spot for successful input and usability
    - Prototype 4 should validate this intuition by adding further controls to see if feedback suggests these additional controls are one step too far

Prototype 4

Taking into account the user feedback from prototype 3, the fourth prototype added in a further element of control in that it allowed the user to stop the recording process and to delete the audio recorded up to that point. This function was added in case the user wanted to start all over again due to severe disfluency.

Following the same procedure as before, using the same five participants under my supervision, I asked each one to follow the sequence of events I provided (below) and to provide feedback about their experience of 1) the control scheme and 2) the ease of use of the ASR system in general compared to the previous iteration.

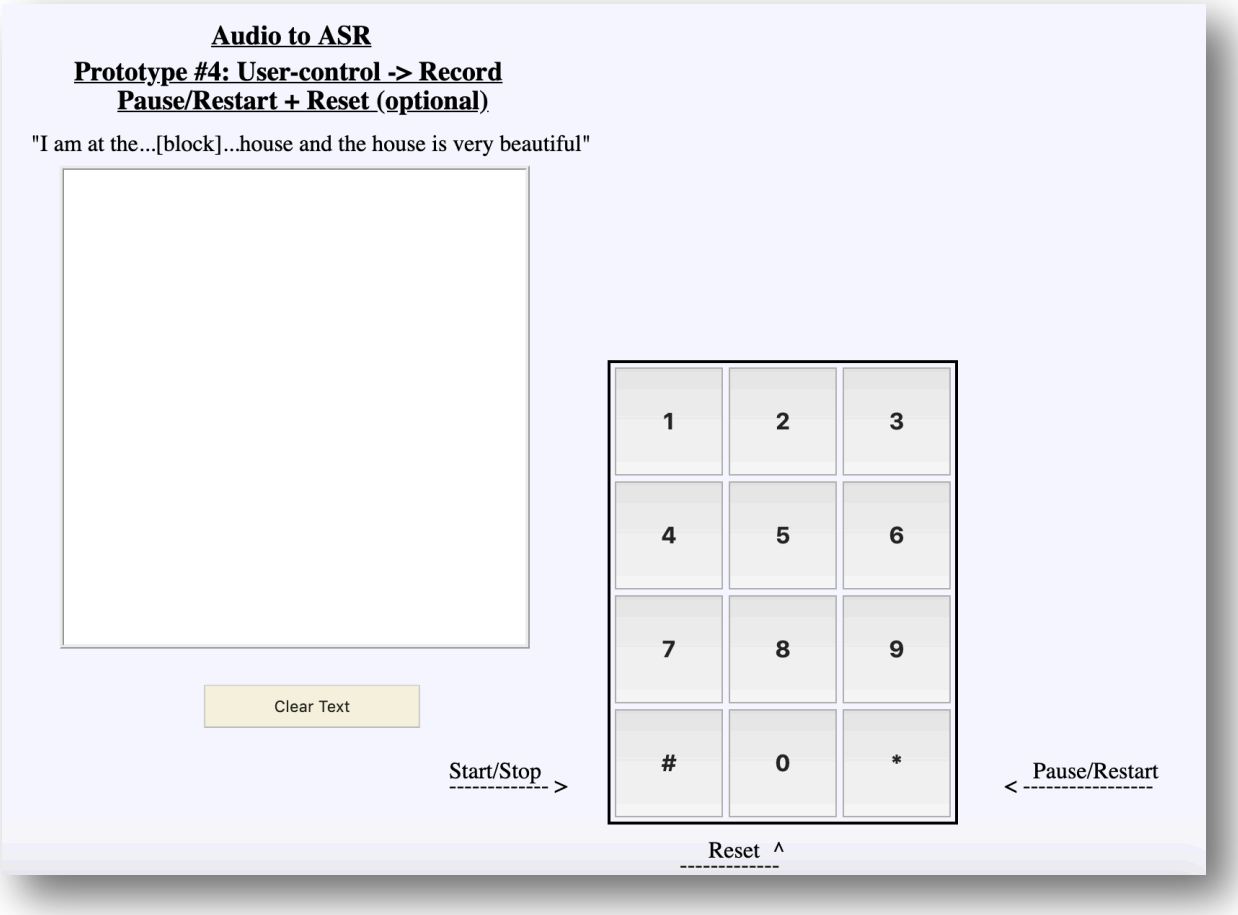


Figure 24 – Prototype 4, recording user speech with additional start/stop, pause/restart, reset

Flow of Events

1. Press the '#' key on the keypad to start the recording process
   [Microphone activated and recording begins]
2. Read the required passage/say what you want

If required:

3. Press the '*' button to pause the recording
4. Press the '*' again to restart the recording

If required:

5. Press the '0' button to stop the recording process and clear the data
6. Press the '#' button to start recording again

Else:

7. Press the '#' button again when finished speaking

[Recording ceases and audio file automatically processed through ASR]

8. Output displayed on screen in the output area
9. Press the ‘clear text’ button to clear the output area

The flow of events for this prototype can be seen in the following diagram:

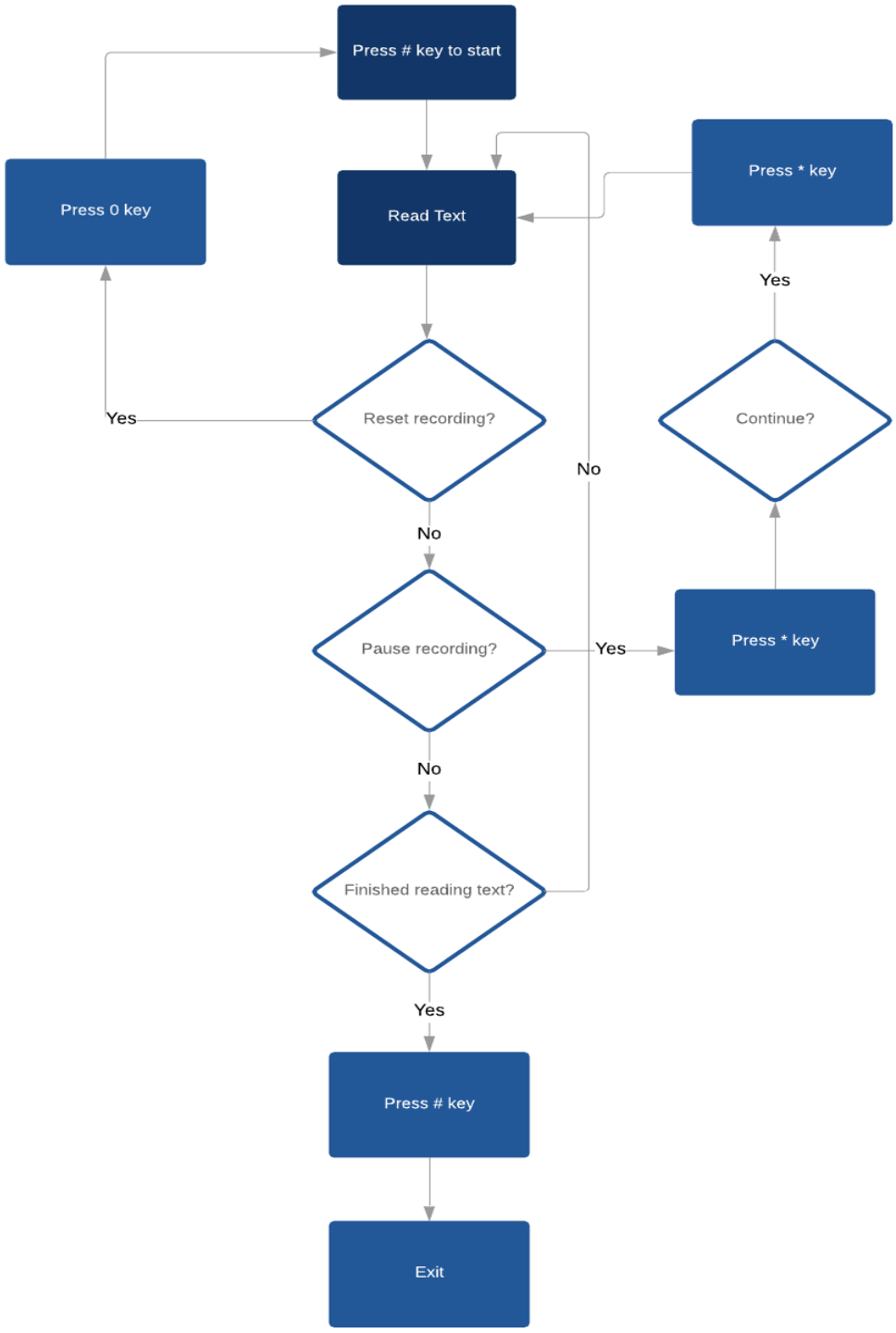


Figure 25 – Prototype 4 flow of events

PARTICIPANT	**Reset Function**	**ASR**
Irfan	"It's a nice idea but I didn't ever feel the need to use it"	"The system as a whole still works well so I don't see what this new option really adds."
Ben	"I didn't want to stop and start all over again. I don't see the point in this feature at all"	"This adds nothing for me."
Patrick	"I would never use this."	"This makes the system harder to use, if I'm honest."
Naheem	"I think it would be easier to just stop the recording and let it throw out its output and then do another recording. It's a lot of effort to reset it."	"Restarting the recording already seems to be built into the system so adding another button just makes it very confusing and harder to use."
Norbert	"The number of options and buttons is now confusing. Having three different buttons is just too much considering I'll be holding the handset to my face too!"	"This adds nothing for me apart from confusion. I'd even go as far as to say it makes it less usable."

Figure 27 – Prototype 4 user testing

Analysis

- Almost all user feedback contained negativity about the number of controls now included in the system
    - All users failed to see the utility in having the 'reset' function
- The usability of the ASR system as a whole was not seen as being negatively affected by the inclusion of the reset function, only the method of input
- Overall, the 'reset' function was seen to "add nothing" to the usability of the system

Overall Analysis and Conclusions

The iterative design and implementation process proved a success because it enabled me to incorporate user feedback into the creation of the prototype designs. The user feedback was used to guide the design process and therefore to end up with a design which maximised the accessibility of the given ASR system for disfluent users.

The goals of this phase of the project were as follows:

- Find a workable solution to the input problem
    - Design a system which was more accessible to disfluent users by removing the following problems:
        - The infinite loop problem

However, one problem still remains

- The audio captured by the ASR system, as modified previously, whilst possibly reducing the number of captured disfluencies, could still contain (potentially long) periods of silence. This data is not useful and will extend the computational processing time of the system

In order to tackle this problem, I perform an amount of signal processing on the audio file before it is processed through the ASR. The function (see Appendix for code description) iterates through the audio data and all data within a certain range of values – considered 'silence' – is removed so that only the 'useful' parts of the audio are retained. This amended file is then put through the ASR.

Figure 28 highlights the effectiveness of this function in reducing the computation time for processing audio files:

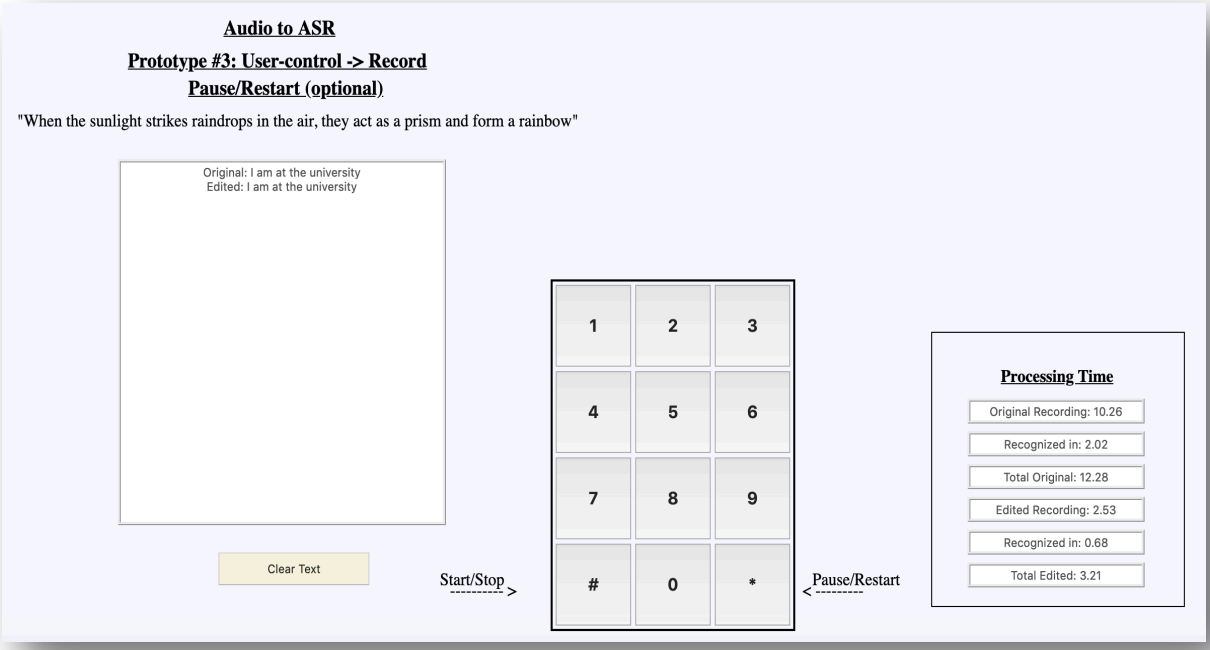


Figure 28 – Example of computational processing time reduction

In the above example, one audio file is processed through the ASR. The utterance contained with the audio is the same and the ASR system recognises this correctly in both cases. The only thing which alters is the processing time:

- File 1
  - Original File
  - Duration: 10.26 seconds
  - Recognition Processing Time: 2.02 seconds
  - Total Processing Time: 12.28 seconds

- File 1
    - Amended File (silence removed)
    - Duration: 2.53 seconds
    - Recognition Processing Time: 0.68 seconds
    - Total Processing Time: 3.21 seconds

As can be seen from this one example, the duration of the audio file is reduced by roughly 75% and the recognition time is reduced by roughly 65%. While this in no way represents the kind of reductions which will be seen generally – the actual reduction in processing time will depend upon how much 'silence' is contained within an audio file – it is clear evidence that the function is effective in reducing the computational processing time to a minimum.

This section has documented the iterative development process which I followed in order to design and implement a telephone-based ASR system which is accessible to disfluent users.

The ‘Output’ Problem

While the system I designed was now fully accessible to the disfluent user and went some way to reducing the number of captured disfluencies, some disfluent speech would still be captured and therefore the following problem remains:

- the captured audio might contain disfluencies and these disfluencies will likely result in mis-recognitions

In order to address this, and because the goal of my project was to investigate methods which did not involve altering the ASR software itself, I decided that a potential method would be to use natural language processing in order to ‘correct’ mis-recognitions. The chosen method takes a corpus as an input. The corpus is then split into bigrams – two-word pairs – as in the following example:

> Corpus: “The cat sat on the mat”
> Bigrams: {'the': ['cat', 'mat'], 'cat': ['sat'], 'sat': ['on'], 'on': ['the']}

This process is completed for all words in the corpus. For the purpose of my project – and in an attempt to increase the accuracy of the ‘correction’ model – I formed bigrams of subsequent words (as above) and also for previous words (as below):

> Corpus: “The cat sat on the mat”
> Bigrams: {'cat': ['the'], 'sat': ['cat'], 'on': ['sat'], 'the': ['on'], 'mat': ['the']}

Following this process, the probability for each subsequent and previous word are assigned, as below:

Subsequent: {'the': {'mat': 0.5, 'cat': 0.5}, 'cat': {'sat': 1.0}, 'sat': {'on': 1.0}, 'on': {'the': 1.0}}

Previous: {'cat': {'the': 1.0}, 'sat': {'cat': 1.0}, 'on': {'sat': 1.0}, 'the': {'on': 1.0}, 'mat': {'the': 1.0}}

As can be seen in this simple example, the probability of ‘mat’ appearing after ‘the’ is 0.5 or 50%; there is a 50-50 chance that, based upon the given corpus, the word ‘mat’ appears after the word ‘the’. Furthermore, the probability of ‘cat’ appearing before ‘sat’ is 1.0 or 100%; based upon this corpus, the word ‘cat’ cannot fail to appear before the word ‘sat’.

Using this model, it is possible, therefore, to make predictions about subsequent and previous words, were we needing to.

For example, if we had the utterance “the cat sunk on the mat” and the word ‘sunk’ had been identified as an error or mis-recognition, then we could use this prediction model in an attempt to find an alternative.

Using the word immediately before ‘sunk’ – i.e. ‘cat’ – the above function will provide a list of candidates for the subsequent word. In this case it is a list of one candidate: ‘sat’, with a

probability of occurrence at 100%. Using the word immediately after ‘sunk’ – i.e. ‘on’ – the above function will provide a list of candidates for the previous word. In this case, again, it is a list of one candidate: ‘sat’, with a probability of occurrence of 100%. The function takes both lists of candidates and compares them for words which occur in both lists in order to create a final candidate list. From this list, the candidate with the highest probability is selected as the best candidate and that word is used to replace the errant word.

Problems occur when the final candidate list contains words which have the same probability value and therefore two or more candidates are selected. In this instance, the function assigns word classes to the candidates – i.e. noun, verb etc. – and uses these to further refine the selection: if ‘sat’ (a verb) had been a candidate to follow the word ‘the’ (a determiner) then this candidate would have been discarded as it breaks the rules of grammar. This system allows for further refinement of the function.

The screenshot below shows this in action:

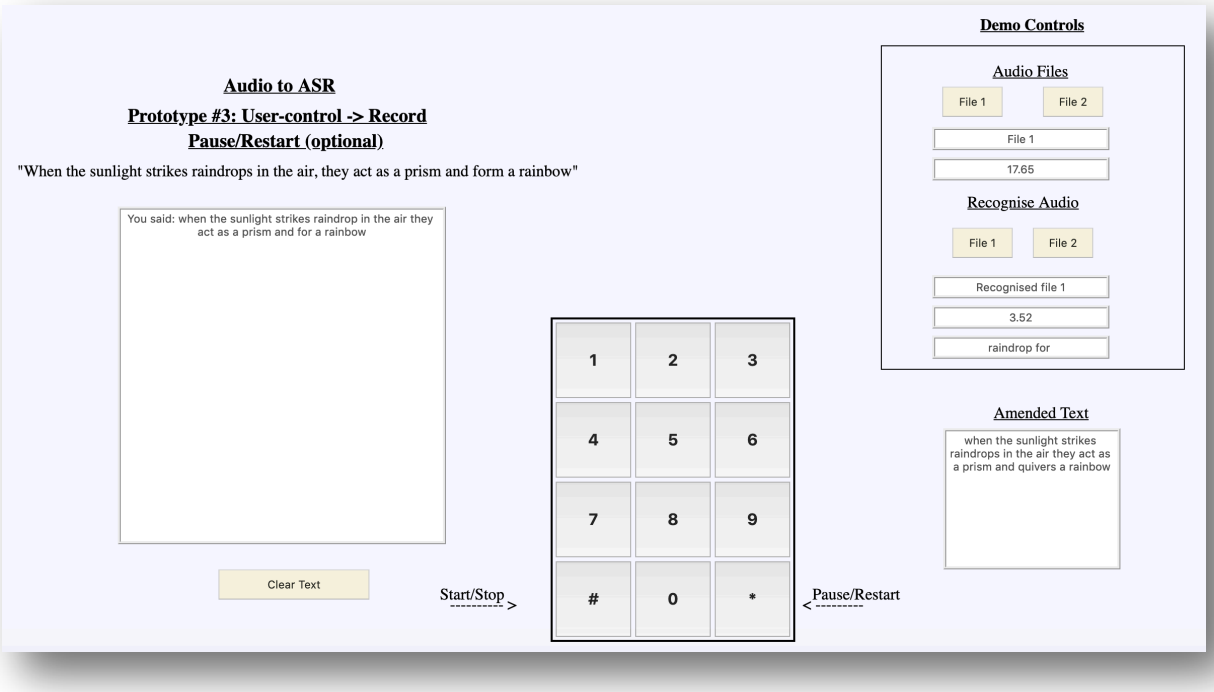


Figure 29 – prediction model in action

In this example, the user has been asked to say the utterance:

> “When the sunlight strikes raindrops in the air, they act as a prism and form a rainbow”

…and the ASR has recognised this, due to disfluencies, as:

> “when the sunlight strikes raindrop in the air they act as a prism and for a rainbow”

As can be seen, the mis-recognitions ‘raindrop’ and ‘for’ have been detected in place of ‘raindrops’ and ‘form’.

Based upon the model described above, these mis-recognitions have been replaced with the candidate words ‘raindrops’ and ‘quivers’. As can be seen, the replacement of ‘raindrop’ with ‘raindrops’ is precisely the result I was seeking, but the replacement of ‘for’ with ‘quivers’ is not. Given the corpus and the probability model derived from it, the word ‘quivers’ is more likely to occur after/before ‘and’ and so was selected for use. In this example, the accuracy of the ASR output was improved by 50%.

Project Evaluation

Successes

As someone totally new to computer science, this was the first time that I had undertaken a project of this scope and complexity. As such, the project management aspects of the project were significant but I'm happy that I was able to incorporate the skills that I've acquired throughout the course – in two software engineering modules – to good effect. For example, key to this project was the development model I chose to utilise and so my acquired knowledge of these allowed me to select a model which was most suitable for the requirements of the project.

From a personal point of view, the requirement to learn three new languages – Python, HTML and CSS – was a significant challenge, which I enjoyed, and was able to reach a level of fluency with all three sufficient to complete the desired goals of the project. I feel that I have developed significantly throughout the course of the project.

In reference to the stated goals of my project, I feel that the 'input' problem – as defined – has been largely solved to a satisfying degree. The resulting system permits the disfluent user to use speech as an input and is presented with no barriers to entry. Success in reference to the 'output' problem is partial and I will discuss this further in the next section.

Challenges

Output Problem

In reference to the 'output' problem, the proposed solution to this problem was only partially successful and had limited application. Whilst the idea to use natural language processing in order to produce a probability model to replace 'incorrect' words was, I felt, a good one, in practise the sparsity of language means that it is very difficult to make it work successfully.

However, with a limited scope, I see no reason why this couldn't work. For example, a organisation who wanted to offer this kind of system (i.e. a telephone-based customer service system) could go about collecting customer service conversation data over a period of time and use that as a corpus. This would provide probabilities between words which are more likely to occur within the context of customer service and would theoretically increase the accuracy of the system.

Furthermore, efficiencies in the processing time could be found if parts of audio considered disfluent could be removed so that the only audio left is 'useful' to the ASR system.

Data Collection Application

On reflection, the data collection protocol was flawed. The protocol I designed provided each participant with four phonetically-balanced scripts and asked them to record themselves reading each one. These recordings were conducted at the participants own discretion and in their own time. The fundamental motivation for this style of data collection was to widen participation so that I was able to collect as much data as possible. Furthermore, I considered that the relative comfort of the collection environment – an environment of the participant's choosing – would also increase the likelihood of wide participation. What I did not consider was that this environmental comfort would result in the audio containing fewer disfluencies than hoped for and, in many cases, none whatsoever. Given the repetition of the data collection task, I would amend the protocol to ensure that more disfluencies were captured – by, perhaps, using a protocol design which made use of spontaneous speech, for example, or perhaps an interview-based protocol.

Future Directions

Any future work would focus upon the output problem and the data collection protocol. I feel that a different approach would be needed for the output problem – machine learning (something currently beyond the scope of my knowledge) would seem to be a potential methodology to investigate in reference to this problem. It might be fruitful to use machine learning to label the features of disfluent speech (when codified as an audio wave) and then to classify it in order to be able to design a system which can identify only these features of the audio; these features can then be manipulated or simply removed to improve the ASR recognition. However, to use such an approach would require a significant amount of training data which can be difficult to obtain as the disfluent users are not always willing to participate in such studies.

Project Conclusion

The stated goals of the project were to design, implement and test a telephone-based ASR system which was accessible to those with disfluent speech, specifically for people who stammer.

In order to meet the requirements of this project, I decided upon an iterative development model in order to best incorporate crucial user testing and feedback; the success of the project hinged upon the feedback provided by users and this model allowed such feedback to be incorporated at the beginning of each iterative cycle. The final system meets the requirements of these goals in terms of accessibility whilst offering a minor improvement for the recognition-related goal.

Bibliography

Alharbi, Simons, Brumfitt, Green (2017). Automatic Recognition of Children's Read Speech for Stuttering Application. 6th International Workshop on Child Computer Interaction http://staffwww.dcs.shef.ac.uk/people/A.Simons/research/workshops/autorecogstuttering.pdf - accessed 28/2/2019

Bloodstein, O. (1975). Stuttering as tension and fragmentation. In J. Eisenson (Ed.), Stuttering: A second symposium (pp. 1–96). New York: Harper & Row.

Brocklehurst, Lickley, Corley (2012). The Influence of Anticipation of Word Misrecognition on the Likelihood of Stuttering. *Journal of Communication Disorders* 45, 147-160

Esmaili, Dabanloo, Vali (2017). An Automatic Prolongation Detection Approach in Continuous Speech with Robustness against Speaking Rate Variations. *Journal of Medical Signals and Sensors* 7(1), 1-7

Overview – Stammering. Accessed 28/2/2019. https://www.nhs.uk/conditions/stammering/

PyAudio 0.2.1 – accessed 28/2/2019 https://pypi.org/project/PyAudio/

PythonAnywhere Home Page – accessed 28/2/2019 https://www.pythonanywhere.com/

SpeechRecognition 3.8.1 – accessed 28/2/2019 https://pypi.org/project/SpeechRecognition/

Standardised Reading Documents – accessed 28/2/2019 https://www.york.ac.uk/media/languageandlinguistics/documents/currentstudents/linguisticresources/Standardised-reading.pdf

Starkweather C. Fluency and Stuttering. Englewood Cliffs, NJ: Prentice-Hall; 1987.

The Future is Voice Search? Spare a thought for those of us with Speech Impediments. Accessed 28/2/2019 http://www.gizmodo.co.uk/2018/03/the-future-is-voice-search-spare-a-thought-for-those-of-us-with-speech-impediments/

The Ultimate Guide to Speech Recognition with Python – accessed 28/2/2019 https://realpython.com/python-speech-recognition/

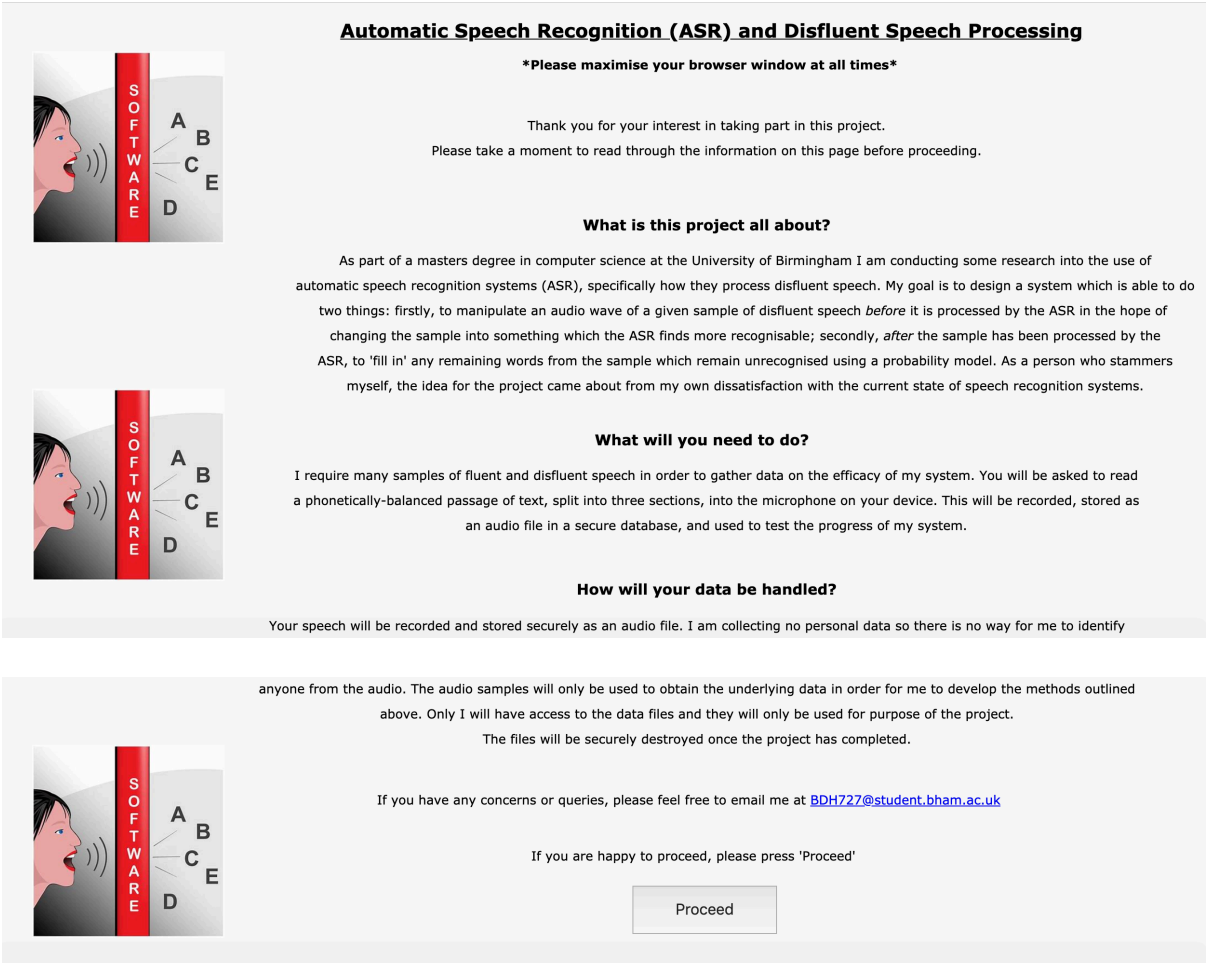
Why Siri won't listen to millions of people with disabilities. Accessed 28/2/2019 https://www.scientificamerican.com/article/why-siri-won-t-listen-to-millions-of-people-with-disabilities/

Web Frameworks for Python – accessed 28/2/2019 https://wiki.python.org/moin/WebFrameworks/

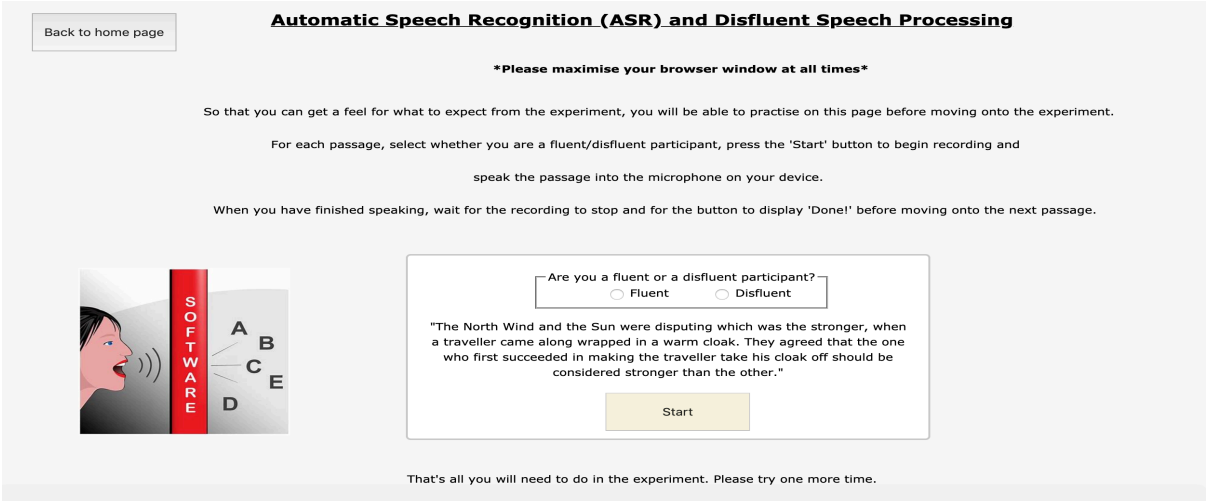
Appendix

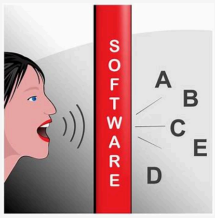
Appendix A – Data Collection Flask Web Application Prototype

Page 1:



Page 2:





Are you a fluent or a disfluent participant?
Fluent Disfluent

"Then the North Wind blew as hard as he could, but the more he blew the more closely did the traveller fold his cloak around him, and at last the North Wind gave up the attempt. Then the Sun shone out warmly, and immediately the traveller took off his cloak. And so the North Wind was obliged to confess that the Sun was the stronger of the two."

Start

If you are happy to proceed and to begin the experiment, please press 'Proceed'.

Proceed

Page 3:

Automatic Speech Recognition (ASR) and Disfluent Speech Processing

Please maximise your browser window at all times

Thank you for proceeding with the project.

You are asked to read several passages of text into the microphone on your device. Your speech will be recorded and stored in a database.

For each passage, select whether you are a fluent or disfluent participant, press the 'Start' button to begin recording and speak the passage into the microphone on your device.

Are you a fluent or a disfluent participant?
Fluent Disfluent

"When the sunlight strikes raindrops in the air, they act as a prism and form a rainbow. The rainbow is a division of white light into many beautiful colors. These take the shape of a long round arch, with its path high above, and its two ends apparently beyond the horizon. There is , according to legend, a boiling pot of gold at one end. People look, but no one ever finds it. When a man looks for something beyond his reach, his friends say he is looking for the pot of gold at the end of the rainbow."

Start

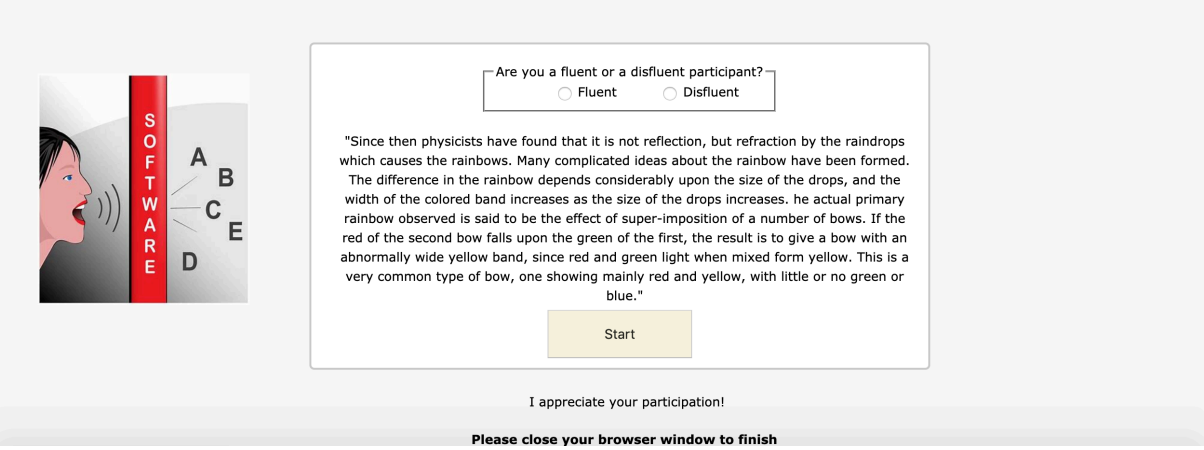
Thanks. When you are ready to begin the next passage, select fluent or disfluent and press the 'Start' button to begin reading the passage.

Are you a fluent or a disfluent participant?
Fluent Disfluent

"Throughout the centuries people have explained the rainbow in various ways. Some have accepted it as a miracle without physical explanation. To the Hebrews it was a token that there would be no more universal floods. The Greeks used to imagine that it was a sign from the gods to foretell war or heavy rain. The Norsemen considered the rainbow as a bridge over which the gods passed from earth to their home in the sky. Others have tried to explain the phenomenon physically. Aristotle thought that the rainbow was caused by reflection of the sun's rays by the rain."

Start

Thanks. When you are ready to begin the next passage, select fluent or disfluent and press the 'Start' button to begin reading the passage.



Appendix B – Deployed Data Collection Web Application

Project Information Page:

ASR and Disfluent Speech Processing

Thank you for your interest in taking part in this project. Please take a moment to read through the information on this page before proceeding.

What is this project about?

As part of my masters degree in computer science at the University of Birmingham, I am conducting research into the use of automatic speech recognition systems (ASR), specifically how they process disfluent speech. As a person who stammers myself, the idea for the project came about from my own dissatisfaction with the current state of speech recognition systems.

If you would like to know a little more about the project, click HERE, otherwise please continue.

What do I need?

I require a range of fluent and disfluent speech samples, in order to analyse and develop methods to assist an ASR system to recognise disfluent speech.

What will you need to do?

You will be asked to read and record four short passages of text into the microphone on your computer. After each recording, the file will be saved to your computer, which you will then upload using the link provided. You are free to record as many or as few passages as you feel comfortable.

How will your data be handled?

Only I will have access to the data files and they will only be used for the purpose of the project. The files will be securely destroyed once the project has completed.

If you are happy to proceed, to begin please press:

HERE if you are a fluent speaker.

HERE if you are a disfluent speaker.

Disclaimer

No personal data is being collected.

(Note: Matt Diamond's Recorder.js is a popular JavaScript library for recording audio in the browser as uncompressed pcm audio in .wav containers. This page was made with this codebase and in no way is ownership claimed.)

Further Information Page:

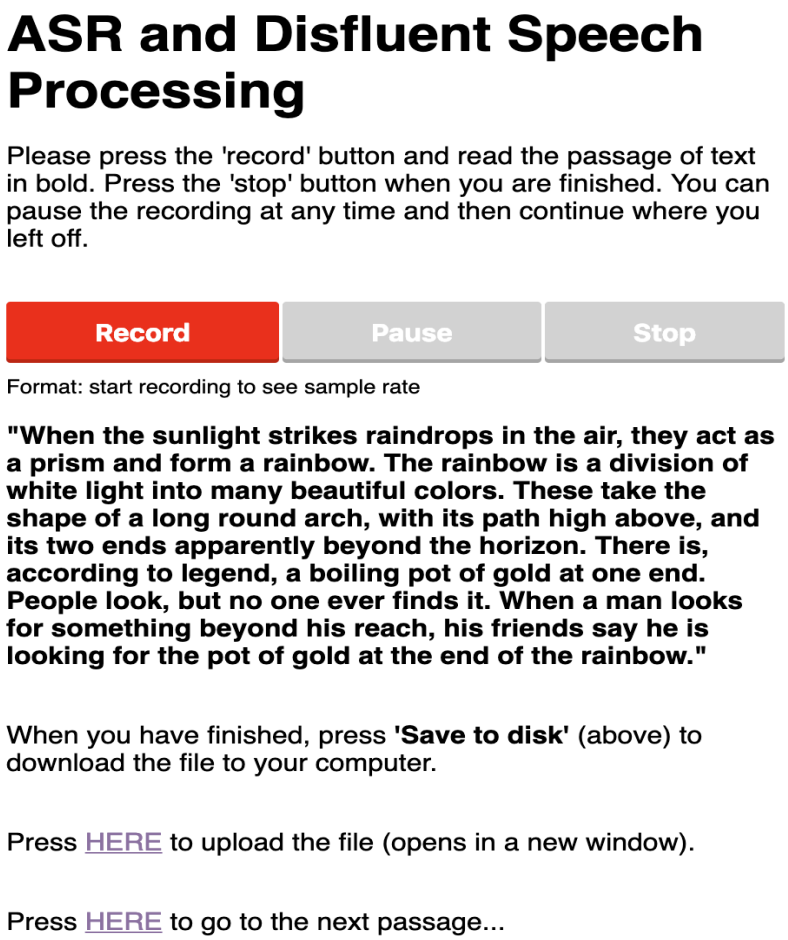
What am I trying to do?

I am conducting some research into the use of automatic speech recognition systems (ASR), specifically how they process disfluent speech. My goal is to design a system which is able to do two things: firstly, to manipulate an audio wave of a given sample of disfluent speech *before* it is processed by the ASR in the hope of changing the sample into something which the ASR finds more recognisable; secondly, *after* the sample has been processed by the ASR, to 'fill in' any remaining words from the sample which remain unrecognised using a probability model. This model will make use of a degree of natural language processing in order to 'predict' which words have not been recognised by the ASR and to fill them in.

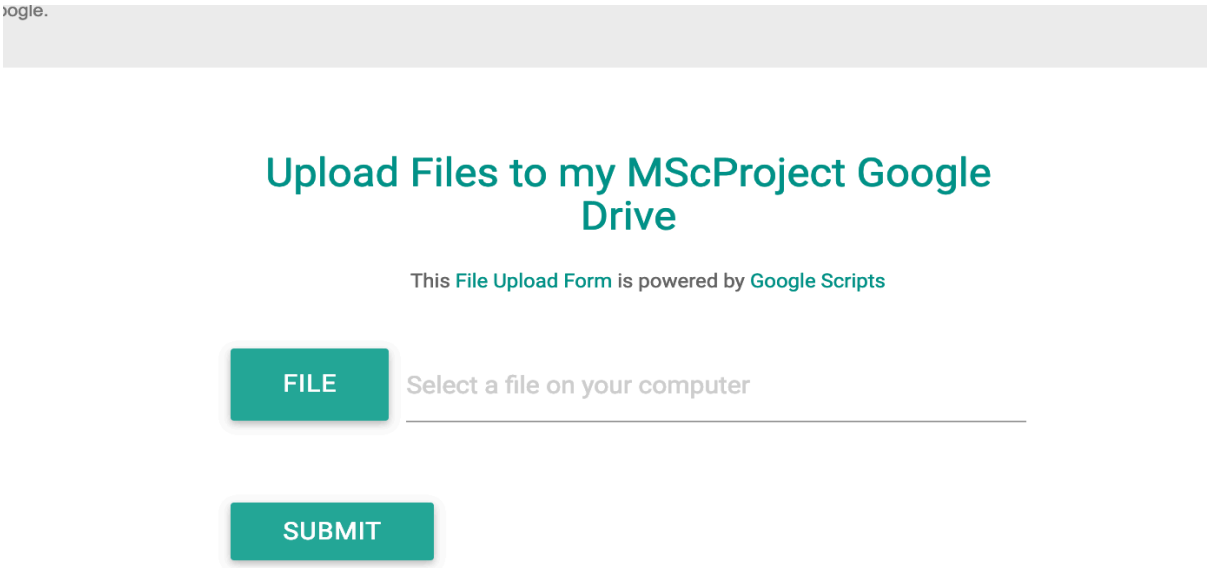
I hope to shed some light upon whether this method is feasible and whether further research would be worthwhile.

Back to the homepage

Data Collection Page 1:



Upload File Page:



Appendix C – Phonetically-Balanced Text Passages

"When the sunlight strikes raindrops in the air, they act as a prism and form a rainbow. The rainbow is a division of white light into many beautiful colors. These take the shape of a long round arch, with its path high above, and its two ends apparently beyond the horizon. There is, according to legend, a boiling pot of gold at one end. People look, but no one ever finds it. When a man looks for something beyond his reach, his friends say he is looking for the pot of gold at the end of the rainbow."

"Throughout the centuries people have explained the rainbow in various ways. Some have accepted it as a miracle without physical explanation. To the Hebrews it was a token that there would be no more universal floods. The Greeks used to imagine that it was a sign from the gods to foretell war or heavy rain. The Norsemen considered the rainbow as a bridge over which the gods passed from earth to their home in the sky. Others have tried to explain the phenomenon physically. Aristotle thought that the rainbow was caused by reflection of the sun's rays by the rain."

"Since then physicists have found that it is not reflection, but refraction by the raindrops which causes the rainbows. Many complicated ideas about the rainbow have been formed. The difference in the rainbow depends considerably upon the size of the drops, and the width of the colored band increases as the size of the drops increases. The actual primary rainbow observed is said to be the effect of super-imposition of a number of bows. If the red of the second bow falls upon the green of the first, the result is to give a bow with an abnormally wide yellow band, since red and green light when mixed form yellow. This is a very common type of bow, one showing mainly red and yellow, with little or no green or blue."

"You wished to know all about my grandfather. Well, he is nearly ninety-three years old; he dresses himself in an ancient black frock coat, usually minus several buttons; yet he still thinks as swiftly as ever. A long, flowing beard clings to his chin, giving those who observe him a pronounced feeling of the utmost respect. When he speaks, his voice is just a bit cracked and quivers a trifle. Twice each day he plays skilfully and with zest upon our small organ. Except in the winter when the ooze or snow or ice prevents, he slowly takes a short walk in the open air each day. We have often urged him to walk more and smoke less, but he always answers, "Banana oil!" Grandfather likes to be modern in his language."

Appendix D – Packages and Functions

Package: ASR_Processing

Description: contains ASR processing code

File: Recognize_Audio.py

Imports:

SpeechRecognition, wave, contextLib, Math, collections. Counter, mysql.connector, re, os

Functions:

get_duration (filename: str) -> float – uses the contextlib and wave module to calculate the duration of a WAV audio file and return as a float.

recognise_audio (filename: str) -> tuple – uses the SpeechRecognition module to recognize the provided WAV file and return as a tuple containing the recognized text and the word count.

calculate_accuracy (filename: str, recognized_text: str) -> tuple – calculates the accuracy of the ASR script compared to the original script. Returns a tuple containing the accuracy rate, the number of mis-recognitions and the mis-recognised words.

process_store_fluent () -> str – uses the calculate_accuracy function to process all audio files in the database classified as being from a fluent participant. Stores the results in the ASR_Results database table. Returns a confirmation string when complete.

Process_store_disfluent () -> str - uses the calculate_accuracy function to process all audio files in the database classified as being from a disfluent participant. Stores the results in the ASR_Results database table. Returns a confirmation string when complete.

File: Calculate_Stats.py

Imports:

mysql.connector, Beautiful Soup

Functions:

get_fluent_stats () -> list – retrieves all data from the database assigned to fluent participants and calculates the required statistics: number of files, number of R1, R2, R3 and G1 files, lowest accuracy rate, highest accuracy rate, average accuracy rate. Returns a list of lists of the results.

get_disfluent_stats () -> list – retrieves all data from the database assigned to disfluent participants and calculates the required statistics: number of files, number of R1, R2, R3 and G1 files, lowest accuracy rate, highest accuracy rate, average accuracy rate. Returns a list of lists of the results.

create_html (fluent_stats, disfluent_stats) -> None – uses the Beautiful Soup module in order to edit the HTML template I wrote – Stats_template.html – and to insert the statistical values provided by the previous two functions.

File: Stats_template.html

HTML template file ready for receiving statistical data.

Package: Audio_Data

Description: Local storage of all participant audio data for more efficient access.

Directories: G1, R1, R2, R3

Sub-directories (for each directory): Disfluent, Fluent

Package: Data_Display

Description: contains code to format data to HTML templates for display.

File: HTML_view_generator_raw.py

Imports: pandas, mysql.connector, Beautiful Soup, re

Functions:

create_view_raw_data () -> None – uses the Beautiful Soup module in order to edit the HTML template I wrote – Raw_audio_data_table.html – and to insert the data values pulled from the database. Makes use of a pandas data frame in order to display data in an orderly fashion and the re module in order to locate positions to edit HTML.

create_view_fluent_data () -> None – uses the Beautiful Soup module in order to edit the HTML template I wrote – Fluent_Audio_Data_Table.html – and to insert the data values pulled from the database. Makes use of a pandas data frame in order to display data in an orderly fashion and the re module in order to locate positions to edit HTML.

create_view_disfluent_data () -> None – uses the Beautiful Soup module in order to edit the HTML template I wrote – Disfluent_Audio_Data_Table.html – and to insert the data values pulled from the database. Makes use of a pandas data frame in order to display data in an orderly fashion and the re module in order to locate positions to edit HTML.

File: HTML_view_generator_asr-processed.py

Imports: pandas, mysql.connector, Beautiful Soup, re

Functions:

create_view_asr_processed_fluent () -> None – uses the Beautiful Soup module in order to edit the HTML template I wrote – ASR_Processed_Fluent_Data.html – and to insert the data values pulled from the database. Makes use of a pandas data frame in order to display data in an orderly fashion and the re module in order to locate positions to edit HTML.

create_view_asr_processed_disfluent () -> None – uses the Beautiful Soup module in order to edit the HTML template I wrote – ASR_Processed_Disfluent_Data.html – and to insert the data values pulled from the database. Makes use of a pandas data frame in order to display data in an orderly fashion and the re module in order to locate positions to edit HTML.

generate_results_pages () -> None - uses the Beautiful Soup module in order to take a String representation of a HTML template insert the ASR result data values pulled from the database and to generate HTML code for each audio file ready for display. All HTML template files are stored in the same package in the RESULTS directory.

Package: Data_Processing

Description: contains code to move data to/from local machine to database.

File: db_to_local_raw.py

Imports: mysql.connector, wave, os

Functions:

from_db_fluent () -> None – Retrieves all files by fluent speakers and filters by script ID in order to store locally in selected directory. Checks for duplicates in local directory to ensure no duplications.

from_db_disfluent () -> None – Retrieves all files by disfluent speakers and filters by script ID in order to store locally in selected directory. Checks for duplicates in local directory to ensure no duplications.

File: local_to_db_raw.py

Imports: mysql.connector, wave, os, scipy.io.wavfile

Functions:

get_filenames_db () -> list – fetches all filenames of audio files currently stored in the database. Returns as a list.

to_db () -> None – Uploads all audio files stored locally to the database. Uses data from get_filenames_db () to ensure that no duplicate files are uploaded.

Package: ASR_Web_Application_Input_Problem_Demo

Description: contains all the code for a standalone Flask-based web application.

File: Run.py

Imports: Flask, render_template, request, pyaudio, wave, numpy, scipy.io, scipy.io.wavfile, os, time, matplotlib.pyplot, Thread, pygame, recognize_speech_problem, recognize_speech_prototype

Global variables:

App -> Instance of Flask object

Utterances -> Dictionary to store ASR results for display to the text area

Timings -> Dictionary to store the time values for calculating computational efficiency

Record -> List containing Boolean value to control audio recording (on/off)

Frames -> List to store audio data frames

Pause_threshold -> List to hold ASR pause threshold values

Functions:

home () – returns a rendered template of Problem.html.

problem_asr1 () – contains code to allow the user of the application to set the ASR pause threshold value – 2, 4 and 6 second values – and to activate the ASR recognition ready for speech input. Calls store_utterance () to store the ASR result in the Utterances dictionary (assigned to a specific key) and calculates the computational processing time before storing the value in the Timings dictionary (assigned to a specific key). Returns a rendered template which pulls data from the Utterances and Timings dictionaries, and calls display_string (), in order to render the data on screen. Also provides the user with the option to clear the text area (where ASR results are displayed) which calls clear_text_area () with an int value specifying the text area to clear.

prototype_page1 () – returns a rendered template of Prototype1.html.

prototype_asr1 () – contains code to allow the user of the application to set the ASR pause threshold value – 5, 10, 15 and 20 second values – and to activate the ASR recognition ready for speech input. Calls store_utterance () to store the ASR result in the Utterances dictionary (assigned to a specific key) and calculates the computational processing time before storing the value in the Timings dictionary (assigned to a specific key). Returns a rendered template which pulls data from the Utterances and Timings dictionaries, and calls display_string (), in order to render the data on screen. Also provides the user with the option to clear the text area (where ASR results are displayed) which calls clear_text_area () with an int value specifying the text area to clear.

prototype_page2 () – returns a rendered template of Prototype2.html.

prototype_asr2 () – contains code to allow the user to record their input which is then processed through the ASR module. User is given options to ‘start’ and ‘stop’ and these inputs, when selected, change the Record variable to True which initiates an implementation of the pyaudio module for recording the input from the user; the ‘stop’ instruction calls stop_recording_asr2 () which changes the Record variable to False and processes the captured audio data through the ASR by calling recognize_speech_prototype (). Two threads are used when these two instructions are called in order to change the text displayed on the HTML buttons to be more informative to the user (i.e. ‘recording’ when the ‘start’ button is pressed).

The ASR result is stored by calling store_utterance () which stores the recognized text to the relevant key in the Utterances dictionary, and the computational processing time is calculated and stored in Timings. Function returns a rendered template of Prototype2.html containing the acquired data from the user. Also provides the user with the option to clear the text area (where ASR results are displayed) which calls clear_text_area () with an int value specifying the text area to clear.

prototype_page3 () – returns a rendered template of Prototype3.html.

prototype_asr3 () – contains code to allow the user to record their input which is then processed through the ASR module. User is given options to 'start' and 'stop' and 'pause' these inputs, when selected, change the Record variable to True which initiates an implementation of the pyaudio module for recording the input from the user; the 'stop' instruction calls stop_recording_asr3 () which changes the Record variable to False and processes the captured audio data through the ASR by calling recognize_speech_prototype (). The 'pause' instruction simply changes the Record variable to false which breaks the loop which records the audio. Two threads are used when these instructions are called in order to change the text displayed on the HTML buttons to be more informative to the user (i.e. 'recording' when the 'start' button is pressed).

prototype_page4 () – returns a rendered template of Prototype4.html.

prototype_asr4 () – contains code to allow the user to record their input which is then processed through the ASR module. User is given options to 'start' and 'stop', 'pause' and 'reset' and these inputs, when selected, change the Record variable to True which initiates an implementation of the pyaudio module for recording the input from the user; the 'stop' instruction calls stop_recording_asr3 () which changes the Record variable to False and processes the captured audio data through the ASR by calling recognize_speech_prototype (). The 'pause' instruction simply changes the Record variable to false which breaks the loop which records the audio, whereas the 'reset' instruction changes the Record variable to False and clears the Frames list of audio data. Two threads are used when these instructions are called in order to change the text displayed on the HTML buttons to be more informative to the user (i.e. 'recording' when the 'start' button is pressed).

results_page () – returns a rendered template of Prototype_Results.html. This page was used for demonstration purposes.

stop_recording_asr2 () – prototype_asr2 () helper method. Used to stop audio recording when initiated by the user. Changes the Record variable to False, processes the audio data through the ASR and stores in the Utterances dictionary assigned to a specific key.

play_audio () – Helper method. Uses the pygame module to play a WAV audio file. Used throughout for demonstration purposes.

stop_recording_asr3 () – prototype_asr3 () helper method. Differs from stop_recording_asr3 () in that it processes the same functionality for prototype 3 and 4. Also calculates additional timing data.

store_utterance (text_area: int, utterance: str) -> None – Helper method for all prototypes. Stores the ASR results in Utterances dictionary assigned to a unique key.

display_string (text_area_no: int) -> str – Helper method for all prototypes. Pulls all data from the Utterances dictionary according to the required text area value and formats into a string which is suitable to be displayed to the screen.

clear_text_area (text_area_no: int) -> None – Helper method for all prototypes. Clears all data from the Utterances and Timings dictionaries so that when data is pulled by the template renderer there is nothing to display.

run () -> None – Launches the Flask application on a specified port.

File: ASR_Processing.py

Imports: SpeechRecognition

Global variables:

Recognizer -> An instance of SpeechRecognition recognizer object

Microphone -> An instance of the SpeechRecognition microphone object

Functions:

recognize_speech_problem (pause_threshold: int) - > str – Takes an int value which represents the desired pause threshold and then launches the ASR into listening mode; this function listens to input taken from the user and returns a String output.

recognize_speech_prototype (file: str) -> str – Helper method for all prototypes. Takes a file name as input and processes said file through ASR. Returns a String result to caller.

Package: ASR_Web_Application_Output_Problem_Demo

Description: contains all the code for a standalone Flask-based web application.

File: Main.py

Imports: Flask, render_template, request, wave, os, pygame, Thread, pyaudio, time, Counter, recognize_speech_prototype, determine_duration, remove_silence, probability_next_word

Global variables:

App -> Instance of Flask object

Utterances -> Dictionary to store ASR results for display to the text area

Timings -> Dictionary to store the time values for calculating computational efficiency

Record -> List containing Boolean value to control audio recording (on/off)

Frames -> List to store audio data frames

Functions:

output1 () – returns a rendered template of Output1.html.

output_asr1 () – codes same functionality as prototype_asr3 (). Used to introduce and demonstrated the 'output' problem.

output2 () – returns a rendered template of Output2.html.

output_asr2 () – uses same base code as output_asr1 () with the following additions. Calculates the computational processing time for an audio file being recorded and processed by the user. Calls the remove_silence () function to remove all data within the audio file considered 'silence'. Calculates the computational processing time for an audio file which has been amended by the removal of all 'silence'.

output3 () – returns a rendered template of Output3.html.

output_asr3 () – *uses* same base code as output_asr1 () with the following additions. Calls the remove_silence () function to remove all data within the audio file considered 'silence'. Uses the probability model described in the 'output problem' section of the report to determine replacement words for all of the mis-recognitions which are detected in the ASR output.

run () -> None – Launches the Flask application on a specified port.

Helper Methods: play_audio (), stop_recording (), store_utterance (), clear_text_area (), display_string (), run (). All have been described in detail earlier in the appendix.

Package: Web_Application_Data_Collection_Prototype

Description: contains all the code for a standalone Flask-based web application.

File: Flask_app_run.py

Imports: Flask, render_template, request, practise_data_process, collection_data_process

Global variables:

App -> Instance of Flask object

Functions:

home_page () – returns a rendered template of Home_Page.html.

practise_page () – returns a rendered template of Practise_Page.html.

collection_page () – returns a rendered template of Collection_Page.html.

practise_record_start_audio1 () - gathers the inputted fluency type from the user. When 'start' button is selected, calls practise_data_process () in order to record the audio from the user and store in the database. Returns a rendered template of Practise_Page_Recording1_Done.html.

practise_record_start_audio2 () - gathers the inputted fluency type from the user. When 'start' button is selected, calls practise_data_process () in order to record the audio from the user and store in the database. Returns a rendered template of Practise_Page_Recording2_Done.html.

collection_start_record_audio1 () - gathers the inputted fluency type from the user. When 'start' button is selected, calls collection_data_process () in order to record the audio from the user and store in the database. Returns a rendered template of Collection_Page_Recording1_Done.html.

collection_start_record_audio2 () - gathers the inputted fluency type from the user. When 'start' button is selected, calls collection_data_process () in order to record the audio from the user and store in the database. Returns a rendered template of Collection_Page_Recording2_Done.html.

collection_start_record_audio3 () - gathers the inputted fluency type from the user. When 'start' button is selected, calls collection_data_process () in order to record the audio from the user and store in the database. Returns a rendered template of Collection_Page_Recording3_Done.html.

run () -> None – Launches the Flask application on a specified port.

File: File_name_generator.py

Imports: random, String

Functions:

file_name_generator (length: int) -> str – generates a random string and concatenates with '.wav' in order to create unique filenames.

File: MScProject.ini

Description: 'ini' file for securely storing the database credentials.

File: Connect_INI.py

Imports: ConfigParser, MySQLConnection, Error

Functions:

read_ini (filename: defaulted to specified path) – uses the ConfigParser module to read the 'ini' file and extract the database credentials.

connect () – calls read_ini () to get the database credentials and opens a connection to the database.

File: Practise_Data_Process.py

Imports: Pyaudio, Connect_INI, file_name_generator

Functions:

process_data (fluency_type: str) -> None – Creates a Pyaudio object and initialises in order to record the audio provided by the user. Stores this audio data, along with the fluency type in the database.

File: Collection_Data_Process.py

Imports: Pyaudio, Connect_INI, file_name_generator

Functions:

process_data (fluency_type: str) -> None – Creates a Pyaudio object and initialises in order to record the audio provided by the user. Stores this audio data, along with the fluency type in the database.

Package: Web_Application_Data_Collection_Deployed

Description: contains all the code for the prototype data collection application along with some additional JavaScript code for handling the audio recording (code is not my own) and a 'yaml' file for configuring the application for deployment on the Google App Engine platform.