Computing BEng

Individual Project

Rubato: An Adaptive Musicality Tutor using Pitch Detection

Conrad Godfrey

Supervisor:
Iain Phillips

June 17, 2014

Abstract

In this project, we propose an adaptive learning web application that employs pitch detection algorithms. As web based learning is growing more and more popular, an increasing number of companies are adopting adaptive learning into their education models, most notably, Duolingo, a language tutoring website. There exists, however, no such application of this type in the domain of music theory, though there are many non-adaptive websites that attempt to teach it. With the advent of HTML5 and the Web Audio API, an opportunity also exists to run browser based pitch detection algorithms to analyse various aspects of a pupil's musical ability, yet there are currently no browser based pitch detection algorithms suitable for measuring vocal performance. We devise a highly robust, fast algorithm suitable for measuring sung pitch in pupils wishing to measure their tuning, and we apply this in an adaptive learning context.

Acknowledgements

I would like to thank my supervisor Iain Phillips for his continued support and guidance throughout my project. I would also like to thank my friends and family for their company for the past three years at Imperial College. A big thank you is also due to those who helped me pitch detect my algorithm, David Maguire, Elise van Lil, Alex Moore, Emily Ooi, and Tom Watkins.

Contents

Chapter 1

Introduction

In this paper I propose an application to harness the power of the modern web browser to deliver engaging musicality tutoring based on adaptive learning principles. The application will comprise of various exercises designed to challenge and improve a user's musicality, and keep track of their progress as well as adapt to their strengths and weaknesses in different areas.

1.1 Adaptive learning

As usage of the world wide web has become ubiquitous among citizens of the developed world, web based teaching has grown rapidly to try and modernize learning methods to fit into the 21st century lifestyle. Increasing numbers of companies like Coursera,[1] Codeacademy,[2] Duolingo[3] are providing easily accessible education for anyone with an internet connection and a desire to learn. Codeacademy and Duolingo in particular are notable for their use of rich, highly interactive learning tools which - through requiring the user to have an input into the educational process - establish a feedback system with powerful results.[4] These companies offer a variety of different programmes the user can study, and each of these is taught through a series of exercises, the results of which are used to track the users progress through the course, and provide statistics on how much they have learned. Duolingo, a website that teaches foreign languages, takes it a step further by introducing adaptive learning methods to recognise the user's proficiency in various areas, information it can then use to tailor-make lessons to fit the user's needs. Duolingo is arguably the best best-known example of an adaptive learning system put into practice. It will therefore often be used for comparison throughout this report, as it has also provided a lot of inspiration for my thinking about the direction my project will take.

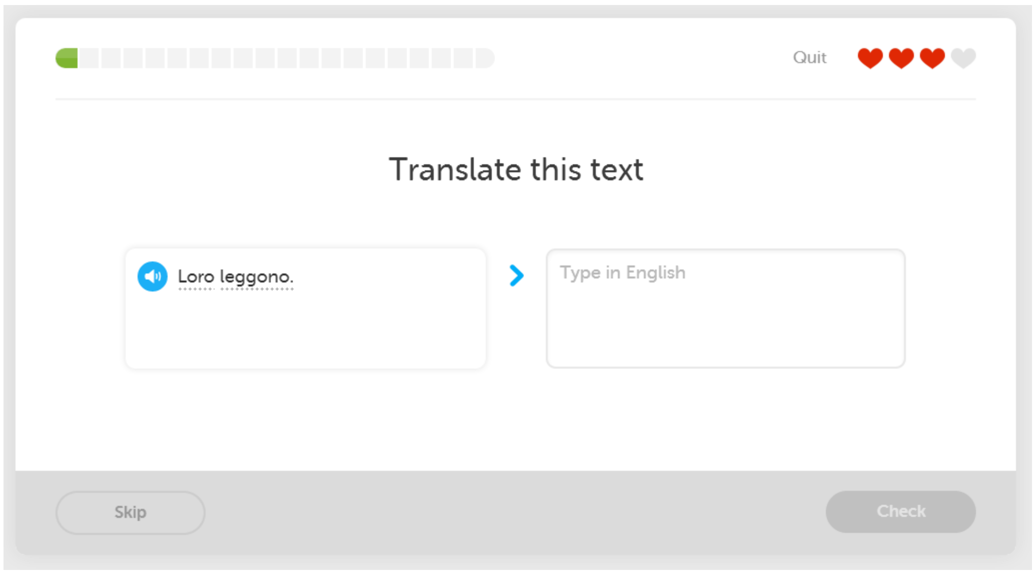


Figure 1.1: Duolingo adapts the questions it asks depending on the user's strengths and weaknesses

1.2 Music teaching on the web

The web is a great place to train the musical ear, a skill we shall refer to under the term of 'musicality' from hereon in. The rich media options possible on modern day web browsers mean that the input and output of music to a web browser is not only easy to implement, but easy to make user friendly. However, while there are many 'music theory tutors' and 'ear trainers' available, there are no existing adaptive learning solutions. Such a solution would hopefully be invaluable to potential learners as the current best solutions still have no idea who you are and what you've achieved after you leave the page.

In order to teach adaptively we must analyse information about a user's performance, and then change how we teach accordingly. There are various ways of measuring musicality, which will be discussed in more detail later, and we will rely on these metrics to adapt the learning experience to fit the user.

1.3 Pitch detection

There are very few examples of pitch detection algorithms running in browser, and the few that there are aren't very useful to a singer.[5,6] This is because they all try to perform real-time pitch detection where the output is calculated at the same time the user is singing. This is useful for instruments like guitars where when you strike a note, you can be assured it will remain fairly constant in pitch, but the human voice is prone to slight variations in pitch that can cause a real-time algorithm to oscillate around a point, making it hard to gauge what the note you actually sung was. It is with this in mind that I aim to create a meaningful pitch detection algorithm that looks at a 2-3 second audio sample and gives the user accurate pitch data for that period

1.4 Achievements

I have created a web application to train user's intonation using pitch detection and adaptive learning techniques. In doing so, I have created a designed a functional adaptive learning method, as well as a successful pitch detection function for the recognition of sung notes.

Chapter 2

Background

In this chapter we cover the theory of adaptive learning, and look at some examples of existing websites that employ adaptive learning. We then look at the basics of Western music theory in terms of pitch, the 12-tone scale, and tuning. Finally, we look at autocorrelation, a method of detecting pitch.

2.1 Adaptive learning

Web based learning has been a topic of interest almost since the birth of the web,[7] and the introduction of the HTML5 standard has opened up even more opportunities for computers to play a role in the education process.[8] 3 notable examples of web based learning systems, a class of web application I refer to as a WLS, are:

- Coursera, a site that provides lecture material for various university level courses.
- Codeacademy, a site designed to help people learn coding by getting them to undertake interactive browser based programming lessons.
- and Duolingo, a site designed to help the user learn foreign languages through interactive exercises that tries to tailor-make each exercise to the users needs by analysing their strengths/weaknesses.

The end aim of these sites is the same as all WLS's: to educate the user, but there are key differences between them which we can use to categorise them further into a hierarchy.

- *static*: This facilitates learning through a pull model, with users viewing learning material on the site, and doing what they like with the information they are given. This only allows a one way interaction with no form of testing/feedback mechanism. Coursera is an example of a static WLS.

- *dynamic*: These build on everything the static WLS has to offer, and include some sort of feedback loop, normally in the form of providing the user with tests/exercises they can use to improve their understanding of the learning materials provided to them. This allows a two way interaction that can help to cement understanding of concepts by allowing the user to put them into practice. Codeacademy is an example of a Dynamic WLS.
- *adaptive*: These includes all the features of a dynamic WLS, only it can treat every user differently by analysing their strengths and weaknesses based on the exercises it supplies them with.[9] This adaptive approach allows for superior teaching to static or dynamic WLS's, as if implemented correctly it will start to mimic the tailor-made learning experience one might receive from a real-world 'for-hire' tutor. Duolingo is an example of an adaptive WLS.

2.1.1 Adaptive learning in music

There are many ways that adaptive learning techniques can be applied in a musical context. As an example, imagine a simple exercise.

- The user is played a rhythmical phrase.
- They must then replicate the phrase by tapping it in on the space bar.
- The application then calculates how accurate the user's approximation of the rhythm is, and feeds this information back to the user

The application can offer the user simple rhythms to start off with, and then, as it notices the user repeat the rhythms with greater and greater accuracy, it can increase the difficulty by introducing syncopation, triplets, and other advanced musical techniques.

2.2 Existing Musicality Tutors

There are a wide variety of existing web-based applications that teach musicality, to varying degrees. There are programs to allow you to practice interval recognition,[10] identify a note in a chord, practice recognition of rhythms.[11] A lot of these programs are standalone, and focussed on one specific area. Musictheory.net[12] is a good example of a website that that goes beyond that, it provides exercises that test a wide variety of skills, trains your ear as well as providing music theory lessons. However Musictheory.net provides no adaption to the user, the furthest it goes is allowing you to specify what you want to be taught, but it is not intelligent enough to work it out itself.

2.3 Music Theory

In order to understand some of the workings of the program it will be necessary to delve briefly into a little music theory. We will examine the fundamentals of Western music theory, as well as theory of pitch. From here on we shall talk exclusively in terms of Western music theory, as it is the theory that the overwhelming majority of contemporary Western music is based upon.

2.3.1 Pitch

The fundamental quality of a note played on the piano, plucked on the guitar, or sung by the voice is its pitch. The other defining quality of a note is its timbre/tone i.e. whether it sounds harsh or mellow, buzzy or clean, but this is a quality that is very hard to quantify, and also not as important to the overall melody of a piece of music - The same melody played on a guitar and a piano can certainly be considered to be the same piece of music, despite the differing tones of the instruments, but if you change the pitch of any notes, it becomes a different melody entirely. So of an individual note in a melody, we can say that its pitch is its defining characteristic.

What do we mean by pitch? Scientifically, the pitch value of an audio signal is determined by that signal's fundamental frequency, f_0, where a higher frequency corresponds to a higher pitch, and a lower frequency a low pitch, however in terms of human perception, it is not as concrete as this, as there are various different psychoacoustic phenomena that determine how 'high' or 'low' a given note sounds. For example, a sinusoidal tone played at the same frequency at a low volume followed by a high volume will appear to be playing two different pitches, with the louder tone sounding lower in pitch. However, sinusoidal tones are much simpler than the rich complex tones of a musical instrument, which due to being constructed of many different frequencies can provide the listener with more cues as to the pitch they are at, so for the purposes of this project we will define pitch as it is defined in most musical contexts, as a logarithmic function of frequency.

Some instruments like pianos and guitars have predetermined pitches that the instrumentalist can produce. Others, like the violin or human voice, can create any pitch within a given spectrum in a continuous fashion. These two classes of instrument are quite different in that the latter type requires a more precise ear to play so that they sound in tune.

The Scale

There are of course an infinite number of pitches, as there are infinitely many different frequencies that an audio signal can have. However, if you start from a base frequency and gradually increase the pitch, you will eventually find that the note 'sounds the same' as the note you started on, only higher. This happens every time you double the frequency, a ratio called an 'octave', and is an interesting property of the ear. Thus in the range of an octave, you can find all the pitches you could possibly think of. For the purposes of making music though, these pitches are quantised into 12 different notes. They are recognisable on the musical keyboard as shown below. The '#' and 'b' characters denote a sharpening or flattening of a pitch, i.e. a raising or lowering of the pitch respectively.

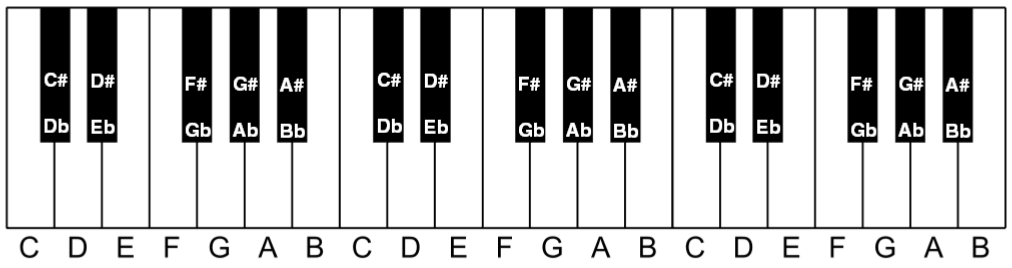


Figure 2.1: The 12 notes of the Western scale. License[1]

Representing pitch

We now know about the 12 different notes in the Western scale, but as we have mentioned, simply knowing that a note's pitch is a 'C' doesn't give us complete information about the note, as we are missing the octave information that tells us how high or low the note is. In this report, there are two main representations we will use.

- **Scientific notation** : This works on the basis that 'middle C', the C in the middle of a normal piano, is represented as C4. The next note above that is C#4 or Db4, followed by D4 and so on until we reach B4, 11 semitones up. The next note after this is the C again, and since we have completed a full octave, we denote it C5. Similarly going downwards, the note directly below C4 is B3 etc. all the way down to C3, where the pattern repeats.

This is arguably the most intuitive way of representing both a pitch and an octave as concisely as possible.

- **MIDI notation** : This is the format used to store MIDI data, the most widely used music sequencing data format. It works on the principle that middle C is represented by the number 60, and incrementing and decrementing this number gives you the notes a semitone either side. This is convenient as it allows us handle the numerical side of musical analysis with ease.

Tuning

Tuning defines the mapping between frequency and notes. Convention dictates that the A above middle C on a piano (A4 in scientific notation or 69 in MIDI) is tuned at 440Hz (this is where the 440 comes from in the equation above), and this is referred to as 'tuning to A440'. In 'Equal Temperament', the tuning system found in most modern pianos, all notes are spaced equally, which means that you can play in any key and still produce the same sound. In equal temperament:

- Doubling the frequency causes an increase in pitch of an octave, so A880 is the next A above A440.
- Since a leap of 12 semitones is caused by a doubling of the f_0 , it follows that the frequency ratio between each semitone (any two adjacent notes on the keyboard above) is $2^{\frac{1}{12}}$ or about 1.059.

Converting from frequency to pitch

For the purposes of this program it will often be necessary to convert between frequency and pitch. Using equal temperament and the MIDI notation, we can define a function thus.

$$p = 69 + 12 \times \log_2 \frac{f}{440Hz}$$

where p is the pitch in MIDI notation, and f is the frequency.

As mentioned previously in the section regarding the scale, there are only 12 different notes that are commonly used, yet this function can provide us with fractional values. What sense can we make of a number like 54.38? The unit most commonly used to represent fractions of a semitone is the cent. There are 100 cents in a semitone, so the number 54.38 would be translated to a F#3 at 38 cents sharp, similarly, a score of 54.8 would be translated to a G that was 20 cents flat.

Intervals

An interval is the distance between two notes as measured by the ratio of their frequencies. We have discussed one interval already, that being the octave, who's frequency ratio is 2:1. The next simplest interval is the 'perfect fifth', who's frequencies have a ratio of 3:2. Now we run into a problem with equal temperament. While the perfectly spaced notes of the piano are useful for their flexibility with respect to what key you wish to play in, they cannot accurately represent the purest intervals such as the perfect fifth. The closest interval on the piano is 7 semitones, or a ratio of $2^{\frac{7}{12}}:1 = 1.4983$. Thankfully however, this slight difference is not great enough to offend most peoples' ears.

The nature of an interval is such that two intervals starting on different notes will have a similar sound. Taken out of any harmonic context, an interval of a perfect fifth starting on an A will have the exact same effect as one starting on a C, or F, or any other note. This is why if you play a piece in a different key to that which it was written (meaning all the notes are shifted down or all are shifted up from where they should be) you can still call it the same piece, and indeed it will still very recognisably be the same piece, as all of the intervals in the accompaniment and melody will be the same).

The importance of good intonation

For instrumentalists of the continuously pitched instruments mentioned above such as the violin and human voice, producing correctly tuned intervals is a vital skill known as 'intonation'. Poor intonation can be the downfall of many a chamber choir, or ensemble, as one person singing wrong notes can lead half the group into another key which will sound very jarringly unpleasant. What's more, good sight reading depends on the ability to not only tune the intervals correctly, but to be able to recall what they sound like, and translate written music directly into performance in real time. Within the classical singing circuit, good sight reading is one of the most desirable, and rare, qualities in a singer, and a good sight reader can command high fees for their time.

Practising intonation is a useful exercise in and of itself however, even for pianists, guitarists and flautists, as a good command of intonation translates to a better musical ear, and can allow you to easily transcribe a melody through recognising the intervals it is comprised of, an exceedingly useful skill for any musician to have. However, there are very few tools to allow you to do, and it is normally only achievable by singing to a well musically trained peer and having them analyse you.

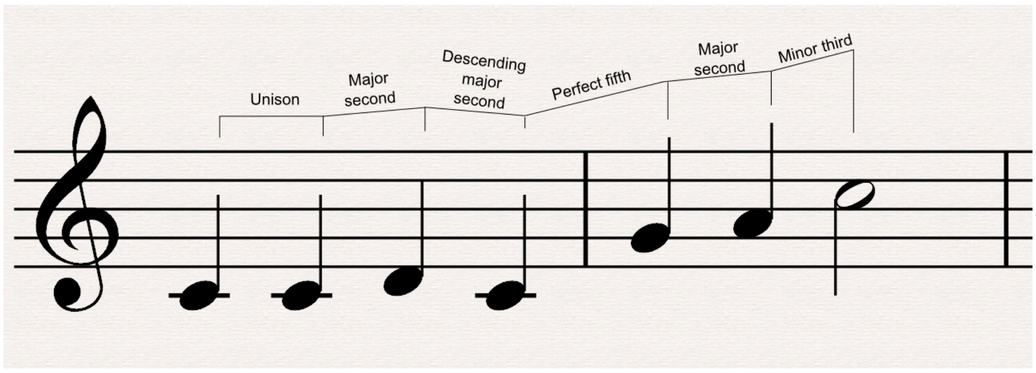


Figure 2.2: Any given melody can be thought of as a sequence of intervals

2.3.2 Human vocal range

Male and female voices can generally be split into 4 types which are determined by their singing range. They are:

- Bass: E2-E4
- Tenor: C3-C5
- Alto: F3-F5
- Soprano: C4-C6

These values will be useful to us when it comes to writing our pitch detection function as they will determine the range of values we have to allow our detected frequency to fall in.

2.4 Pitch Detection

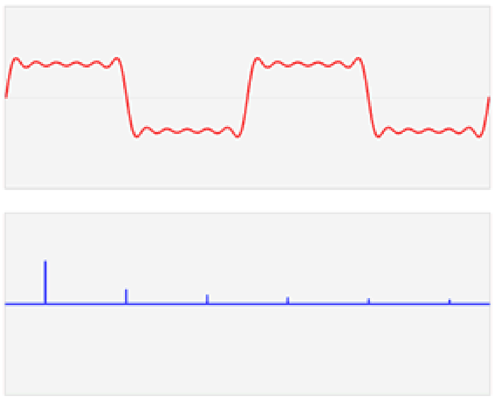
Pitch detection is a well understood problem that has been researched for many years. The pitch of a note, as described above, is the human perception of how high or low it is. The pitch value of an audio signal is determined by the fundamental frequency, f_0 of the signal, and thus the problem of pitch detection of a signal is analogous to finding that signal's fundamental frequency.

2.4.1 Fundamental Frequency Detection Methods

f_0 detection is a difficult process, and there is no 'best method' so to speak, each approach has its drawbacks and advantages, the normal trade-off being that a method that is fast may not be reliable and vice versa. There are two main approaches to f_0 detection, analysing the signal in the the time-domain, or the frequency domain.

To analyse in the frequency domain requires the use of the Fourier Transform, and is therefore a slower process, but provides more accuracy, as well as support for multiple notes being detected simultaneously. However, as we are only using monophonic pitch detection, and responsiveness is key, then for our purposes, time-domain algorithms will be the most appropriate.

Figure 2.3: Time domain (above) vs frequency domain representations of the same signal, the Time domain representation plots amplitude against time, and is a quantisation of the raw audio signal. The second image represents the frequencies present in the audio signal, and shows their intensity on the y-axis



Autocorrelation

A standard method of time-domain f_0 detection is autocorrelation.[13] It exploits the fact that a periodic or quasiperiodic waveform such as a sustained sung note will be self-similar by the definition of periodicity. If we compare a waveform with a copy of that waveform offset by the period of the waveform i.e. ${f_0}^{-1}$ then we should expect to see a strong correlation between them. So autocorrelation works by iterating over all the possible offset values, and determining which one has the best correlation. This correlation value for each different offset is described by the equation below, where x is the signal function, N is the window size of the waveform you are considering, and v is the offset value.

$$R_x(v) = \sum_{n=0}^{N-1-v} x[n]x[n+v]$$

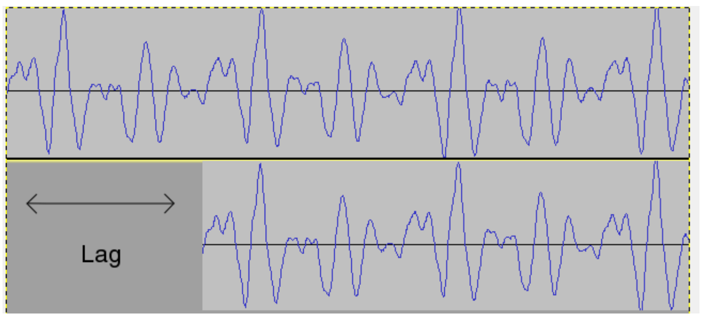


Figure 2.4: This lag value will have high autocorrelation

Limitations

One fundamental limitation of Autocorrelation is that octave errors are frequently encountered. This occurs when the offset is calculated to be double what it should be, and is prone to happening as the nature of a periodic wave is such that if you shift it along twice the offset then you will also end up with a valid offset.

2.5 Audio processing

An analog audio signal can be represented digitally as a series of "samples" These samples are quantised versions of the amplitude of the audio signal.

2.5.1 Nyquist-Shannon sampling theory

The Nyquist-Shannon sampling theory states that to be able to represent a periodic signal with frequency B, we must sample at a rate at least twice that of B. In terms of our pitch detection, the highest frequency we should need to measure will be no higher than 1000Hz, requiring a sampling rate of 2000Hz. Since most microphone capture audio at 44.1KHz or more, then we will have a more than adequate sampling rate.

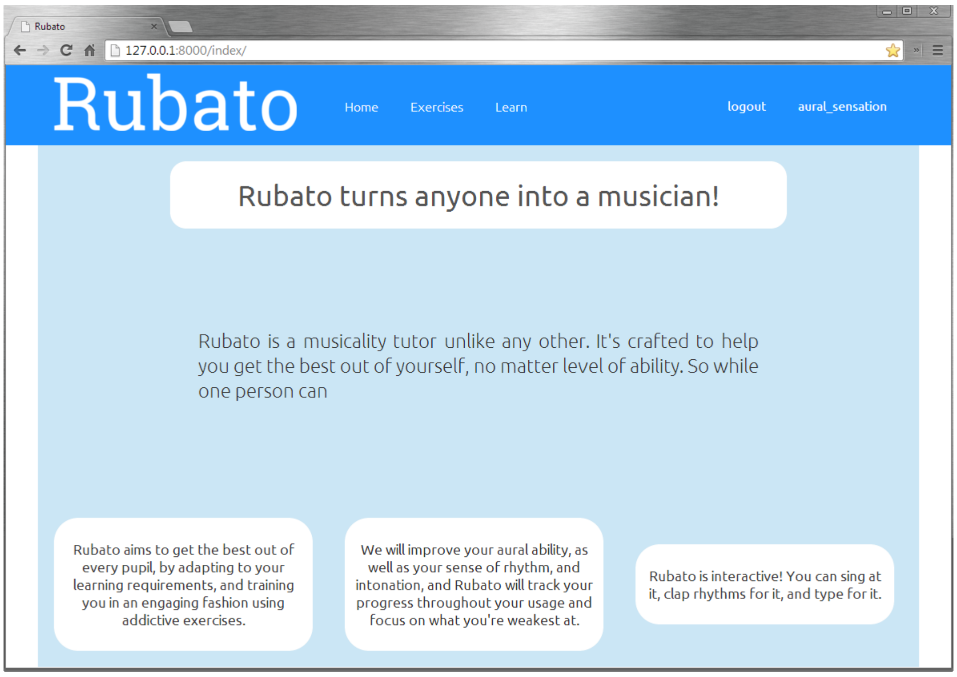
2.5.2 Pulse Code Modulation

Pulse Code Modulation (PCM) is the most common format of storing uncompressed digital audio data, and is an array of quantised samples. In our application, we are using 32 bit PCM, where each sample of audio is represented as a 32 Bit Floats between -1 and 1.

Chapter 3

Implementation

This chapter discusses the implementation of adaptive learning and pitch detection techniques in the form of a web application. I use a variety of software libraries and frameworks to facilitate this, and these are mentioned, as well as a justification of my platform choice. The use implementation of a segmented autocorrelation algorithm to handle pitch detection is discussed, as well as key error correction ideas. Further to this, we discuss the server's handling of user data and its adaptive algorithms.



Rubato turns anyone into a musician!

Rubato is a musicality tutor unlike any other. It's crafted to help you get the best out of yourself, no matter level of ability. So while one person can

Rubato aims to get the best out of every pupil, by adapting to your learning requirements, and training you in an engaging fashion using addictive exercises.

We will improve your aural ability, as well as your sense of rhythm, and intonation, and Rubato will track your progress throughout your usage and focus on what you're weakest at.

Rubato is interactive! You can sing at it, clap rhythms for it, and type for it.

Figure 3.1: The home page of the website

3.1 Platform choice

As the modern internet browser has become more and more powerful, web apps have been able to reduce the previous speed disadvantages they faced when compared with their native equivalents. The web-platform is advantageous due to having a singular codebase, meaning that it can be used by anyone with a web browser, independent of their device, so it has the potential to reach more users. Another key advantage of building a web app is that as user testing is so key to evaluating the app's success, when the time comes, and I want to show it to users, I can easily point people to my website and people will know how to get to it. Distributing a mobile app to others for testing purposes is painful, and difficult to update once they have installed it, with a web app I can instantly change the build of my website, and whoever is testing it won't have to do anything more than refresh the page to get the latest changes.

3.1.1 Mobile browsing

While the application is not available is a native Android or iOS application, on Android, it can be run in browser as Google Chrome supports the Web Audio API. Unfortunately, support of this library is limited on iOS, and it can only support audio playback, and not audio capture from a microphone source.

3.2 Application stack

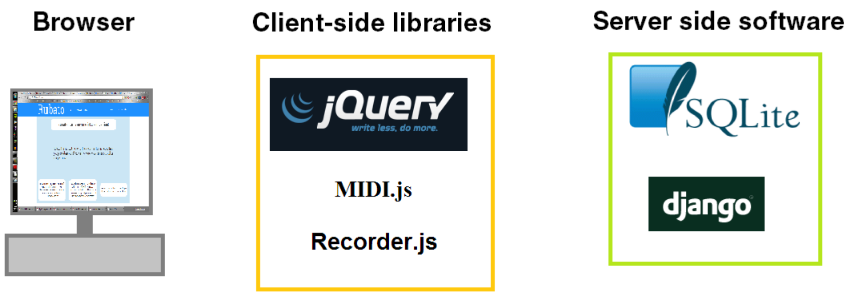


Figure 3.2: The 2-tier application stack for Rubato, showing client and server side technologies used

3.3 Server-side software

3.3.1 Django web framework

The Django web framework[14] allows web developers to create complex web applications quickly, and with little fuss. It is based on the Python language which is known for its emphasis on readability and speed of development. It also includes a simple web server for development purposes, and this is what we have used throughout the project. If this application was ever deployed widely, then we would need to consider using a proper web server like Apache or Nginx, both of which have django bindings, because the Django development server does not have the necessary security checks to ensure it is not prone to malicious attacks. This doesn't really affect the behaviour of the web app, so using the development server makes more sense, as it easy to configure, run, and restart.

3.3.2 SQLite

SQLite is a self contained, serverless database that comes preconfigured to use with Django. Django Models (discussed later in this chapter) essentially deal with all database activity for us, so it is unimportant to know what goes on inside the SQLite database.

3.4 Client side JavaScript libraries

We use multiple Javascript libraries to deal with common tasks like recording audio, audio playback, and asynchronous communication with the server.

3.4.1 Recorder.js and the Web Audio API

Recorder.js and the Web Audio API allow us to seamlessly capture audio in the browser. To start with, we must create an AudioContext.

```
var mediaStreamSource = new webkitAudioContext();
```

If we want to any sort of audio work in the browser, this is the object that the Web Audio API requires us to use. It represents a series of audio modules linked together that form some sort of audio processing pathway. For the purpose of capturing audio from the microphone, we need to create a MediaStreamSourceNode() within the context.

```
var mediaStreamSource = context.createMediaStreamSource(s);
```

Once we've done this, Recorder.js comes into play. It is simply a matter of running the following command

```
recorder = new Recorder(mediaStreamSource);
  recorder.record();
```

and we can start recording media. When we want to stop we can use

```
recorder.stop();
```

3.4.2 MIDI.js

MIDI.js provides us with a way to synthesise audio playback in browser. Once we've set up the plugin, we simply have to call the following function to play a note of our choosing

```
  function playNote(){
      var delay = 0; // play one note every quarter second
      var note = MIDI_note; // the note's pitch in MIDI notation
      var velocity = 127; // how hard the note hits
      // play the note
      MIDI.setVolume(0, 127);
      MIDI.noteOn(0, note, velocity, delay);
      MIDI.noteOff(0, note, delay + 0.75);
  }
```

3.4.3 JQuery

The JQuery library enhances the functionality of vanilla HTML and JavaScript by providing various utility methods/simplifying existing methods. We use it to dynamically update elements on the page, and importantly, to communicate with Django via AJAX (Asynchronous JavaScript and XML). The idea of AJAX is to be able to communicate to the server without reloading the page, and we will use it to facilitate adaptive learning, which will be discussed further in Section 3.11.

3.5 Pitch Detection

Pitch detection is an important part of the exercises, and as such, robust pitch detection has been an important feature in the development of this program.

Most existing pitch detection algorithms are based on real-time applications, i.e. where audio captured from a microphone is analysed and the pitch is displayed as a constantly updating function of the audio signal. This is useful for applications like guitar tuning, where you need near instant feedback, and you can be sure that the pitch you are inputting is fairly constant.

With analysis of pitch in the human voice however, the problem is that even with well trained singers, the pitch of a note can vary quite considerable over a 2 second period, due to vibrato[1], or other causes that can be hard to pick up just by listening to yourself, so it is not always trivial to determine one pitch to summarise a sample of a single held note, and indeed, often not appropriate to do so if the singer cannot even stay on the same note for that short period of time, and wobbles around the note inconsistently.

It is with this in mind that I have developed a Segmented Autocorrelation Algorithm, that can accurately determine the pitch of a 2-3 second sample of singing.

3.5.1 Segmented Autocorrelation Algorithm

The principle of the Segmented Autocorrelation Algorithm is relatively simple: A given audio sample is split into several segments, pitch detection is run on each segment, and then the segments are analysed to calculate pitch. I have also employed several error detection/correction methods that are detailed below.

3.5.2 Breaking it apart

The first step in the algorithm is to segment the audio signal, this allows us to individually analyse the pitch of each segment, so that if the pitch varies, we can detect it.

Choosing a segment size

Most PC microphones sample audio at either 44,100Hz or 48,000Hz, as these ensure that the full 20KHz spectrum of human hearing can be recorded due to the NyquistShannon sampling theorem (see Background Section 2.5). For the sake of simplicity we will assume a sample rate of 48,000Hz for this report, though the program can handle either rate.

The human vocal range defined in Section 2.4.2 was E2 - C6, which corresponds to a frequency range of 80-1000hz.[15] To calculate the number of samples at either end of the range:

$$48000 \div 96 = 600 \text{ samples}$$

$$48000 \div 1000 = 48 \text{ samples}$$

[1]Vibrato is the musical technique of intentionally oscillating the pitch of the note around a fixed point.

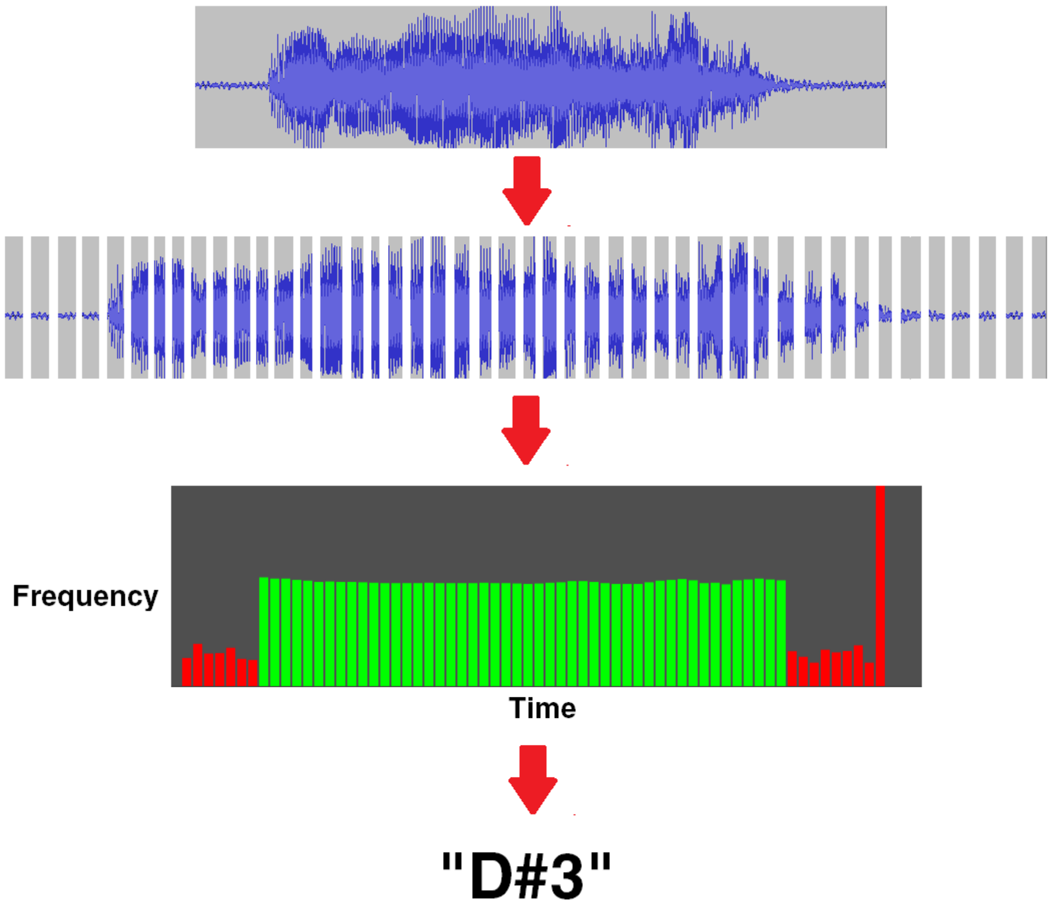


Figure 3.3: An illustration Segmented Autocorrelation. In the first image, we have the original source audio stored in uncompressed PCM format, with time on the x and amplitude on the y, in the next image it is broken into several equally sized segments of length 1200 samples, and then each of these segments is analysed using autocorrelation to give a frequency reading, shown on the third panel. Red portions represent data discarded as junk by the algorithm
. These are then analysed to give a final pitch reading for the given audio sample.

In order for our autocorrelation algorithm to work, it is necessary to take a segment size at least twice that of the lowest frequency we wish to sample. This is because in autocorrelation, we must compare the signal with an offset version of itself, and this score is maximal with an offset equal to the period of the signal. Therefore, in our example, if we use anything less than a 1200 sample size and are trying to detect a 96Hz signal, we won't even be able to autocorrelate on a full period. I have therefore decided to use 1200 as the segment size. Increasing this number results in a higher quality analysis per segment, but the trade-off is that segments per second decreases, meaning you are sacrificing granularity of change of pitch over time.

Autocorrelation

The algorithm for the pitch analysis for individual segments is the autocorrelation algorithm outlined in the first chapter. It is fast, simple to implement, and we don't mind so much about errors due to the fact that it's not a real-time implementation. This means we can look at each segment in the context of the whole recording, and run error correction algorithms outlined below if we do encounter any dodgy pitch readings.

```
function autoCorrelate(buf, sampleRate){

    var MIN_SAMPLES = 48; // corresponds to ~1000 Hz  i.e. C6
    var MAX_SAMPLES = 600; // corresponds to ~80hz i.e. E2
    var SIZE = segment_size; //1200
    var best_offset = 0;
    var best_correlation = 0;
    ACF = new Array(MAX_SAMPLES-MIN_SAMPLES+1).
    for (var offset = MIN_SAMPLES; offset <= MAX_SAMPLES; offset++)
        {
        var correlation = 0;
        var max = SIZE-offset;

        for (var n=0; n<max; n++) {
            correlation += (buf[n])*(buf[n+offset]);
        }

        ACF[offset-MIN_SAMPLES] = correlation

        if (correlation > best_correlation) {
            best_correlation = correlation;
            best_offset = offset;
        }
    }
```

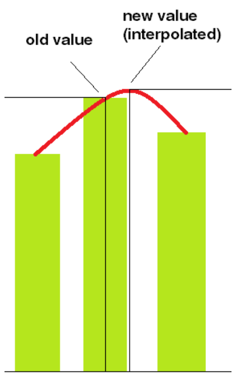


Figure 3.4: If we simply use the peak of our autocorrelation function to determine the pitch, we limit ourselves to detecting pitches only of discrete intervals. Using quadratic interpolation, we can get a better estimate of the best autocorrelation value

```

    return sampleRate/best_offset
```

Enhancing the results: Quadratic Interpolation

With autocorrelation, one inherent limitation is the granularity of the detected pitch. Let's say a we are trying to identify a sung D4. This has a frequency of 293.66Hz and will therefore correspond to a lag of $48,000Hz/293.66Hz = 163.45$samples. Our algorithm at present does not have a way of dealing with fractions of a sample, so at best we will detect 163 or 164 samples. This lack of precision in this case will cause an error of roughly ±5cents, which isn't too bad. The problem however becomes more significant as we detect notes with higher frequencies, as these notes will correspond to smaller lag times, and thus a discrepancy in lag detected will correspond to a more noticeable error in pitch detection.

One way to solve this problem is quadratic interpolation.[16] The idea is to take three points and built a guadratic curve between them. By looking at the maximum of this curve, we can interpolate and find a new peak that will be more accurate.

The equation to calculate the quadratic interpolation for three different point $x_1 < x_2 < x_3$ is as follows:

$$x_{\max} = \frac{1}{2}\frac{B_{23}f(x_1) + B_{31}f(x_2) + B_{12}f(x_3)}{d_{23}f(x_1) + d_{31}f(x_2) + d_{12}f(x_3}$$

where:

$$B_{ij} = i^2 - j^2$$

$$d_{ij} = i - j$$

we then simply take our $x_{\max}$ and make that into our new offset.

3.5.3 Detecting errors

There are two primary methods of detecting errors after autocorrelation has been performed. Firstly, we detemine the active area of the recording, this being the section of the recording where the user is actually singing. To do this, we apply a filtering method based on the RMS Amplitude of the sample. Secondly, we apply a median filter to the sample to smooth out any clearly anomalous pitch readings detected during the active area.

Median Filtering

Median filtering is a noise reduction process used when sharp spikes occur in a signal. It works using a sliding window algorithm, applying a median filter to all the elements in a grid. The code below describes an implementation in Javascript, and Figure 3.2 gives a visualisation of how it works. For this application, the median filter is more appropriate than the more standard moving average filter, as it is non-linear meaning that large spikes will be completely removed, not just smoothed over.

```javascript
function medianFilter(signal,window_size){
    var medians = new Array(signal.length);
    for (var i = mid; i<signal.length-mid; i++){
        var mid = Math.floor(window_size/2);
        var startIndex = i - mid;
        var median_window = new Array(windown_size);
        for (var j = 0; j <window_size; j++){
            median_window[j] = signal[startIndex + j];
            }
        }
        medians[i] = median(median_window);
    }
    return medians;
}
```

In my implementation I use window size 3, as large spikes of this sort are fairly rare, and when they do occur they're usually only 1 segment wide.

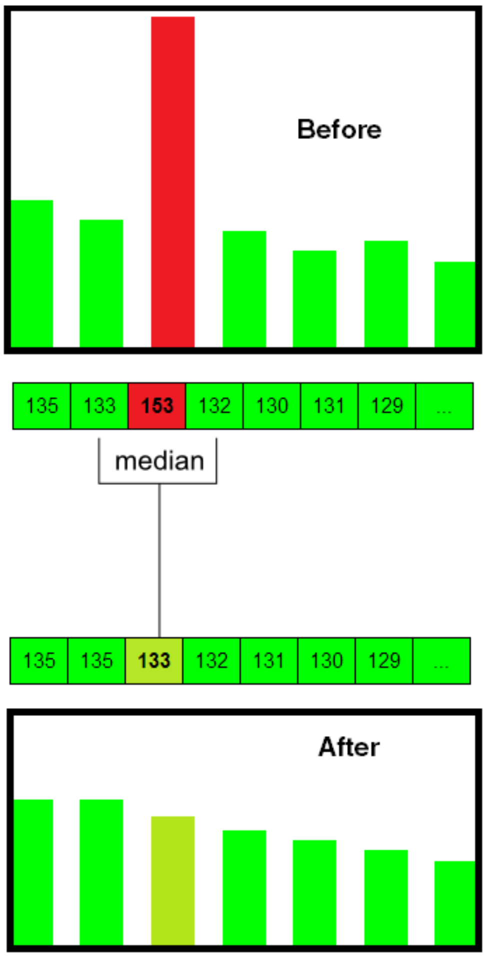


Figure 3.5: A visualisation of the median filter with window size 3: Each point is transformed to the median of a sample size 3 centered on the point. The red column of height 153 is therefore transformed to median(133,152,132) = 133

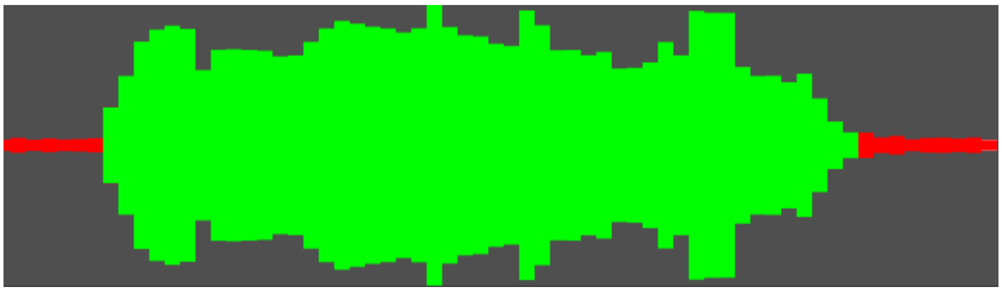


Figure 3.6: A recording of a sung note showing amplitude vs time. The red areas have been filtered our with amplitude filtering

Amplitude filtering

A typical audio sample will have junk data at either end of the recording as the user prepares to sing after activating the microphone, and then leaves the microphone running for a fraction of a second after singing. Amplitude filtering is designed to detect where the edges of the useful data are, and cut them off.

During the autocorrelation, as well as calculating the pitch for each segment, we also calculate its amplitude using root mean square (RMS). It is preferable to use RMS to the conventional peak amplitude method normally used on audio signals, as our window size is sufficiently high that a spike in a given segment due to external noise could cause the amplitude value to be too high. RMS should avoid this as it takes an average over a given sample. The RMS amplitude is calculated using the following algorithm:

$$\sqrt{\sum_{i=1}^{n} s_i^2/n}$$

where n is the number of samples in the segment, and s_i is the i-th sample in the segment. After analysing all segments we end up with a result similar to that shown in Figure 3.3, with a block of loudness surrounded by two blocks of quietness.

Microphone calibration To make sense of this, we have to work out what data should be accepted, and what should be rejected as too quiet to contain valid pitch data. This is done through calibration: The first time the user opens an exercise that uses microphone data, the program records a 2 second sample and segments it similarly to the preprocessing of the autocorrelation method. The minimum amplitude of all segments is then taken as the calibration value, a_0. When determining which segments to cut off in the amplitude filtering stage, we discard all segments with amplitude $< 2a_0$. This value has been determined empirically to be a reasonable threshold for minimum amplitude

3.6 Putting it together

Now that we've determined a suitable chunk of frequencies that represent the pitch of note, we need to work out how to combine them into one reading.

Rolling standard deviation

In order to ensure that large contiguous regions of mispredicted pitch that cannot be fixed with median filtering do not throw of our pitch detection , we need to separate the frequencies into regions of fairly close pitch. We do by grouping frequencies into contiguous regions with a standard deviation below a certain threshold.

Once we have done this, we look for the largest contiguous region in our sample, and calculate pitch by taking the average frequency in the region. If the largest contiguous region is smaller than 60% of the size of all our valid frequency data, we throw an error, and ask the user to try again as we cannot choose an authoritative region.

3.7 Measuring user ability

User ability is measured through a level system. Every user has a level value assigned to each exercise they attempt, and an overall level also. Users with a high level will not have to start from the beginning when attempting new exercises. All users start at at level 1 when they sign up. To increase their level, the user has to complete exercises correctly. As their level increases, the exercises adapt and get harder. Measuring user ability is achieved by assigning a score to every 'attempt' at a certain exercise. These scores are then processed by the server to determine:

- Whether the user should be levelled up.
- What questions it is best to ask them next.

3.7.1 Intonation training

The scoring system for intonation training is as follows:

$$\text{difference} = |P_{\text{actual}} - P_{\text{target}}|$$

$$\text{perfect\_score} = 0.1$$

$$\text{distance\_from\_perfect} = \max(\text{difference} - \text{perfect\_score}, 0)$$

$$\text{score} = \max(1 - \text{distance\_from\_perfect}, 0)$$

Intuitively, this means that any interval attempt less than 0.1 semitones from the actual pitch is classed as a perfect score, and anything over 1.1 semitones away is classed as a zero,with values between those 0.1 and 1.1 semitones determined by linear interpolation.

Every time the user completes an attempt at an interval, their score is calculated and sent to the server. In order to ensure that the exercise is not interrupted, we use Asynchronous Javascript and XML (AJAX) which allows the user to continue playing while their score is communicated to the server in the background. The JQuery library contains support AJAX, and we use it like this.

```
function sendScore(){
    var score = calculateScore();
    var i = interval;
    $.post('/interval/send_score', {score: score.toString(),
       interval: i.toString()}, function(data){
     });
}
```

This simple piece of code sends the score to the server, as well as the interval the score was calculated for.

3.8 Adapting to user ability

When the server recieves the score, we store it in a database, so that we can retain a history of all their past attempts. Django interfaces with the database through Models. An example of this is the IntervalScore model below. This particular model stores all the information about a user's attempt at singing an interval in the interval training exercise.

```
    class IntervalScore(models.Model):
        interval = models.ForeignKey(Interval)
        timestamp = models.DateTimeField(auto_now_add=True)
        score = models.FloatField();
        user = models.PositiveIntegerField();
        def __str__(self):
            return " Score: " + str(self.score) + " for" + self.
               interval.name + " with user id: " + str(self.user)
```

The interval field represents the interval that the user has attempted, the timestamp tells us when it was attempted, and the score field tells us the calculated score for that attempt.

To ensure that the user is tested with an interval appropriate for their skill level, every time the user requires a new interval to attempt, they must request it from the server. The server then obtains their interval score history by querying the database with the User's ID.

Depending on the level of the user, as well as their history, a different interval will be sent.

- If the user's last 3 attempts at a given interval have all been below a certain threshold (calculated using the level of the user), the user will be sent that interval, as well as a prompt to playback the interval and listen to what it sounds like before attempting it agin.
- Different intervals are classified into levels, and you can only unlock some of the harder musical intervals (e.g. a major 7th, diminished 5th) once you reach a certain level
- Your level is incremented by 0.025 for every musical interval you get correct, and decremented by 0.01 for interval gotten wrong.

Chapter 4

Evaluation

In order to evaluate the success of my program, I have used both qualitative and quantitative methods which I will outline in the chapter.

4.1 Adaptivity

How well I have achieved adaptivity in my program has been a difficult thing to evaluate, it is hard to quantify how well someone has been taught, and to achieve a sample size of students anywhere near what you would need to meaningful perform quantitative analysis of their performance has been difficult. I have, however managed to ask a few musical, and non-musical volunteers to use the program and give me their feedback.

Among more musically experienced volunteers, the feedback was good. Users reported they felt challenged by the exercises, and they noticed when the program increased or decreased it's difficulty.

For the less musically experienced volunteers, the feedback was less positive. Some of the starting levels of the exercises were reported to be too difficult, and they felt like they didn't know what was meant to be expected of them.

4.2 Pitch Detection

Robiner, Cheng, Rosenberg, and McGonegal's work in analysing pitch detectors for speech analysis give several different criteria for evaluation of a pitch detector,[13] and we are going to use 3 of them to evaluate ours. They are:

1. Accuracy in estimating pitch period

2. Robustness of measurements. Does the pitch detector hold up in different

environments other than the one it was developed in? Different voice types, and microphones could have an impact on this.

3. Speed of operation

4.2.1 Period accuracy

To test the period accuracy of the detector it will be necessary to compare it's output to a source with a known period. I have not been able to find any existing banks of data that I can use for this, so I have come up with my own.[17]

The test set consists of many different samples of 1 or 2 seconds of a note being sung or played on the piano at a known pitch. I have included samples from multiple different people in my test set to make sure my pitch detection works for as many different voice types as possible. I have also recorded people using a very basic microphone inbuilt on my laptop, so I can ensure that the pitch detection will work even with fairly low quality audio.

The vocal test set includes 70 samples of both male and female voice singing a range of different notes The piano test set includes 36 samples of notes being played over a 3 octave range, from a G2 to G5. A link is available in the bibliography to a zip file containing all of these samples in WAV format.[17]

For both the piano samples and the vocal samples, the pitch detector was 100% accurate in detecting the pitch to the nearest semitone. It is difficult to say with any greater degree of certainty how accurate the voice sample detection accuracy was, as we can't be entirely sure whether it is the voice that is very subtly singing the wrong note, or the detector detecting the wrong period. For the piano however, we can examine the results in greater detail, as we can be more certain of the correctness of the pitch. The results are seen in table below. Firstly, we can calculate the sample standard deviation of the cents given by the following formula:

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x}^2)}$$

Where N is the the sample size, and $\bar{x}$ is the mean of the samples. This gives us a value of 10.180. To put this into perspective, the human ear is estimated to be able to distinguish differences of pitch of about 5-6 cents,[18] and a semitone is 100 cents, so this is an acceptable range for the standard deviation.

Interestingly, the mean of this data is 3.081 cents, implying that on average, our pitch detection is a bit sharp. It is unknown whether this is a function of some acoustical phenomenon on the piano, or whether this is a problem with all detection. The pitch detector has been used to detect computer generated sine waves as well, which it achieved with very high precision, and no sharp/flat bias, so we can rule out some systematic mathematical bias caused by incautious rounding of a variable, or something like that.

Piano results

Expected pitch	Detected pitch	Cent error
G2	G2	10
Ab2	Ab2	4
A2	A2	5
Bb2	Bb2	6
B2	B2	-3
C3	C3	1
C#3	C#3	5
D3	D3	7
Eb3	Eb3	4
E3	E3	4
F3	F3	2
F#3	F#3	-12
G3	G3	-5
Ab3	Ab3	3
A3	A3	3
Bb3	Bb3	9
B3	B3	4
C4	C4	6
C#4	C#4	7
D4	D4	8
Eb4	Eb4	4
E4	E4	1
F4	F4	5
F#4	F#4	1
G4	G4	3
Ab4	Ab4	3
A4	A4	-7
Bb4	Bb4	0
B4	B4	4
C5	C5	2
C#5	C#5	6
D5	D5	10
Eb5	Eb5	32
E5	E5	23
F5	F5	5
F#5	F#5	-8
G5	G5	-38

4.2.2 Robustness

The pitch detector was tested in various different environments, such as on different laptops, and in environments with high background noise. As the autocorrelation is quite noise tolerant, we didn't find a significant change in performance when tested with high background noise.

4.2.3 Speed

The speed of the pitch detector was evaluated over the test set, and determined to be 0.207 seconds per sample. This is fast enough to not cause the user to wait around, so this is a satisfactory measurement for our purposes.

Chapter 5

Conclusions

This chapter summarises the achievement of this project, as well as pointing out ideas for future work.

Adaptive learning in the context of music education is an unexplored field, as adaptive learning is still a growing field itself. In this project, I have devised a computer application that provides an adaptive learning experience for intonation training using a segmented autocorrelation algorithm. Initially, I hoped to include several different exercises that would test a wide variety of musical skills, however, I have only managed to get the intonation exercise up and running. The exercise itself is, I believe, a successful one, and one unlike anything else existing. Adaptive learning has been successfully implemented in this exercise, and when a user spends time on it, they will find that it changes to match their ability. The pitch detection algorithm is highly successful and reliable, and it has shown that it can cope with variable levels of background noise.

5.1 Future

Though the project is finished, there is much more that could be done to enhance and extend it.

- A greater variety of exercises. At present, the only exercise available to the user is interval training. There are many different exercises that could work in an adaptive learning context, for example, the rhythmical exercise mentioned in the background.
- A melody analyser. The pitch detection algorithm is nearly sufficiently sophisticated as to be able to detect pitch in samples containing melodies. This would be an interesting feature that could pave the way for automated music dictation.

Bibliography

[1] "Coursera, take the world's best courses, online, for free.." `http://www.coursera.org/`.

[2] "Codeacademy, learn to code interactively, for free.." `http://www.codeacademy.com/`.

[3] "Duolingo, free language education for the world!." `http://www.duolingo.com/`.

[4] R. Vesselinov and J. Grego, "Duolingo effectiveness study," *City University of New York, USA*, 2012.

[5] "Javascript based real-time pitch detection.." `http://webaudiodemos.appspot.com/pitchdetect/index.html`,.

[6] "Flash based real-time pitch detection.." `http://www.audiostretch.com/pitch/`,.

[7] S. Alexander, "Teaching and learning on the world wide web," in *Proceedings of AusWeb*, vol. 95, p. 93, 1995.

[8] L. S. Griffiths, R. Ogden, and R. Aspin, "A profile of the future: what could html 5 do for he by 2015?," *Research in Learning Technology*, vol. 20, 2012.

[9] "Duolingos data-driven approach to education." `http://blog.duolingo.com/post/41960192602/duolingos-data-driven-approach-to-education`. Accessed: 2014-02-20.

[10] "Interval ear trainer - recognise the interval between two notes.." `http://www.8notes.com/school/theory/interval_ear_trainer.asp`,.

[11] "The rhythm trainer - rhythm recognition and representation.." `http://www.therhythmtrainer.com/`,.

[12] "musictheory.net, on online music theory tutor and ear trainer." `http://www.musictheory.net/`.

[13] L. Rabiner, M. Cheng, A. Rosenberg, and C. McGonegal, "A comparative performance study of several pitch detection algorithms," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 24, pp. 399–418, Oct 1976.

[14] "Django project homepage." `https://www.djangoproject.com/`,.

[15] "Table of note frequencies.." `http://en.wikipedia.org/wiki/Scientific_pitch_notation#Table_of_note_frequencies`,.

[16] "Quadratic interpolation formula." `http://www-rohan.sdsu.edu/doc/matlab/toolbox/optim/tutori5b.html`,.

[17] "Test set for pitch detection algorithms." `http://www.tiny.cc/zdnlhx`,.

[18] D. B. Loeffler, "Instrument timbres and pitch estimation in polyphonic music," 2006.