Voice recognition

Identifying a Voice Print Using the Fourier Transform

Team Leader: BUZATU Rareș Tudor

BĂIAȘU Elena Monica

CIOCAN Alina Cristina

DUMACHI Andreea Ionela

MATEI Constantin Adrian

SPIRIDON Ionuț Nicușor

WINDISCH Adrian Alexandru

Supervisor: Professor, PhD Mihai CARAMIHAI

Table of Contents

Abstract

Speaker Recognition is the methodology through which the deliverer of the speech, diction or audio is detected. The process consists of identifying a particular speaker from a pool of speakers whose **voice samples** have already been stored inside a **database**.

The usefulness of identifying a person from the characteristics of his voice is increasing with the growing importance of automatic information processing and telecommunications. In this paper, we provide a brief overview of the area of speaker recognition, describing its system, various modules of feature extraction and modelling, underlying techniques and some indications of performance. Moreover, a voice recognition system that would identify different users based on previously stored voice samples is to be presented. To achieve this, the speaker's voice sample is compared with the pre-recorded samples already stored in the database prior to the verification process. The results from applying the **Fast Fourier Transform** is compared with the pre-sampled audio stored in the database and if it is returned as a match, then the particular speaker's name is displayed.

Introduction

Everywhere around us are signals that can be analyzed. Signals are time-varying quantities which carry a lot of information. They may be audio signals, images or video signals, sonar signals or ultrasound, biological signals such as the electrical pulses from the heart, communications signals, or many other types.

In the last few years, several numbers of security systems based on fingerprints, voice, iris and face images have been presented. The security system based on iris or fingerprint is not convenient in practice since it imposes stringent requirements on interaction with the user; the users of modern fingerprint may become anxious about the hygiene of the process, and the users of modern iris face stringent requirements on movements and visibility of the eyes, which is why people wearing glasses could feel discriminated. So there is a strong motivation to work with a security system based on voice.

Currently, along with efforts to develop computer procedures that understand spoken messages, there is also considerable interest in developing procedures that identify people from their voices. Being able to speak to your personal computer, and have it recognize and understand what you say would provide a comfortable and natural form of communication.

Over the past few years, it has become increasingly more popular to use voice recognition for applications. Applications such as Siri, Cortana or Google Assistant are able to transcribe audio and understand what a user is saying. Some of these applications are also able to uniquely identify the individuals. At the same time, applications such as Shazam and projects like MusicBrainz have been able to identify music using only 2-3 seconds of recorded music. Audio processing has traditionally been too computationally expensive to be able to be done locally. However, that is changing as new processing algorithms are being created.

Speaker recognition is similar to voice recognition. Voice recognition is to identify *what* is being spoken or uttered, whereas speaker recognition is to identify *who* is delivering the speech or audio. Speaker recognition can be broadly classified into identification and verification. It is a process of identifying and then verifying who is speaking on the basis of patterns extracted from someone's own voice. These patterns may be the features such as the pitch, amplitude or frequency variations present in the voice. Speaker identification is the process of determining which registered speaker provides a given utterance (one-to-many matching), on the other hand, speaker verification is the process of accepting or rejecting the identity claim of a speaker (one-to-one matching), and this decision depends on the degree of similarity between the claimed speaker and the enrolled speaker.

In brief, two basic operations are carried out here: feature extraction and classification. In feature extraction, the vital characteristic which makes it possible to distinguish a speaker from the other is captured. The main technique used for extracting features was Fast Fourier Transform (FFT). We will go deeper into detail on the following pages.

Fourier Analysis

The Discrete Fourier Transform (DFT) is one of the most powerful tools in digital signal processing. The DFT enables us to conveniently analyze and design systems in frequency domain; however, part of the versatility of the DFT arises from the fact that there are efficient algorithms to calculate the DFT of a sequence. A class of these algorithms are called the Fast Fourier Transform (FFT). This article will firstly review the computational complexity of directly calculating the DFT and then it will present how a class of FFT algorithms, i.e., decimation in time FFT algorithms, significantly reduces the number of calculations.

The Cooley–Tukey algorithm, named after J.W. Cooley and John Tukey, is the most common fast Fourier transform (FFT) algorithm. It re-expresses the discrete Fourier transform (DFT) of anarbitrary composite size $\mathbf{N = N_1 N_2}$ in terms of $\mathbf{N_1}$ smaller DFTs of sizes $\mathbf{N_2}$, recursively, to reduce the computation time to $\mathbf{O(N \log N)}$ for highly composite **N** (smooth numbers). Because of the algorithm's importance, specific variants and implementation styles have become known by their own names, as described below. The algorithm, along with its recursive application, was invented by Carl Friedrich Gauss. Cooley and Tukey independently rediscovered and popularized it 160 years later.

Computational Complexity of Directly Calculating the DFT

The N-point DFT equation for a finite-duration sequence, x(n), is given by:

$$\mathbf{X(k)=\sum_{n=0}^{N-1} x\,(n) e^{-j2\pi Nkn}}$$

A complex addition itself requires two real additions. Therefore, each DFT coefficient requires $\mathbf{4N}$ real multiplications and $\mathbf{2N+2(N-1)=4N-2}$ real additions. To have all the DFT coefficients, an N-point DFT analysis requires $\mathbf{4N^2}$ real multiplications and $\mathbf{N(4N-2)}$ real additions. The important point is that the number of calculations of an N-point DFT is proportional to $\mathbf{N^2}$ and this can rapidly increase with N. Figure 1 plots the number of real multiplications versus the DFT length N.

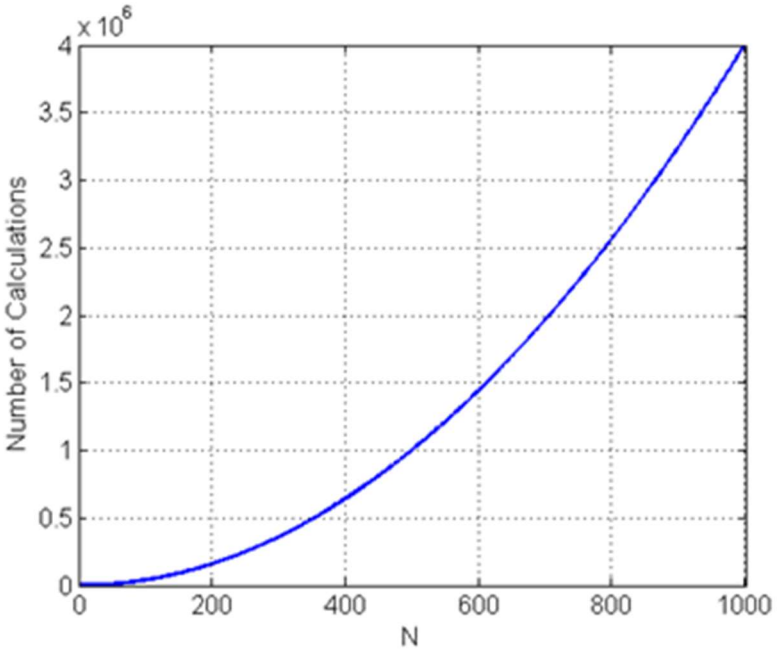


Figure 1. The number of real multiplications for an N-point DFT

Investigation of an Eight-Point DFT

Assume that **x(n)** is a sequence of length eight:

$$\mathbf{X(k) = x(0)e^{-j2\pi k\times 0/8} + x(1)e^{-j2\pi k\times 1/8} + x(2)e^{-j2\pi k\times 2/8} + \cdots + x(7)e^{-j2\pi k\times 7/8}}$$

We see that the length of the DFT, $\mathbf{N = 8}$ in this case, appears as the denominator in the complex exponentials. This leads us to the idea that if we choose and group some terms in a way that allows us to simplify the fractions $-\frac{j2\pi kn}{8}$, we have

$$\mathbf{G(k) = x(0)e^{-j2\pi k\times 0/4} + x(2)e^{-j2\pi k\times 1/4} + x(4)e^{-j2\pi k\times 2/4} + x(6)e^{-j2\pi k\times 3/4}}$$

The remaining terms, which correspond to odd-index samples, are given by

$$\mathbf{H_1(k) = x(1)e^{-j2\pi k\times 1/8} + x(3)e^{-j2\pi k\times 3/8} + x(5)e^{-j2\pi k\times 5/8} + x(7)e^{-j2\pi k\times 7/8}}$$

To simplify the fractions $-\frac{j2\pi kn}{8}$, we can simply factor $\mathbf{e^{-j2\pi k/8}}$ and obtain

$$\mathbf{H_1(k) = e^{-j2\pi k\times 1/8}(x(1)e^{-j2\pi k\times 0/4} + x(3)e^{-j2\pi k\times 1/4} + x(5)e^{-j2\pi k\times 2/4} + x(7)e^{-j2\pi k\times 3/4})}$$

Defining: $\mathbf{H(k) = x(1)e^{-j2\pi k\times 0/4} + x(3)e^{-j2\pi k\times 1/4} + x(5)e^{-j2\pi k\times 2/4} + x(7)e^{-j2\pi k\times 3/4}}$ we get the final result as $\mathbf{X(k) = G(k) + e^{-j2\pi k\times 1/8}H(k)}$, where **G(k)** and **H(k)** correspond to four-point DFTs.

Since **G(k)** and **H(k)** are periodic with 4,

$$\mathbf{X(k+4) = G(k) + e^{-j2\pi(k+4)\times 1/8}H(k)\ .}$$

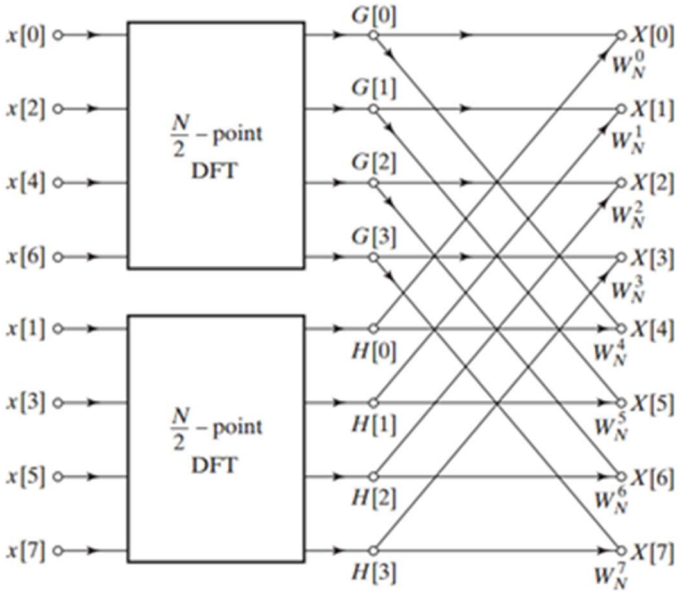


Figure 2. The flow graph of breaking an eight-point DFT into two four-point ones. Image courtesy of *Discrete-Time Signal Processing*

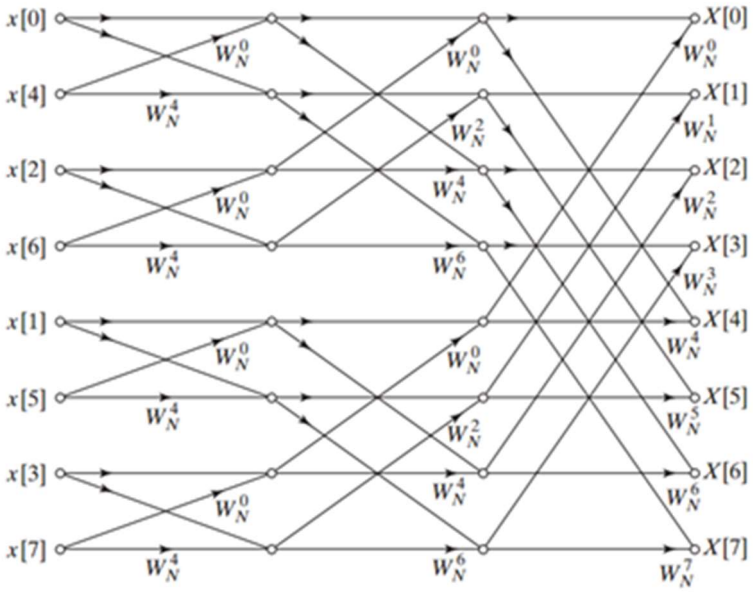


Figure 3. The decimation-in-time based structure to compute an eight-point DFT. Image courtesy of *Discrete-Time Signal Processing*

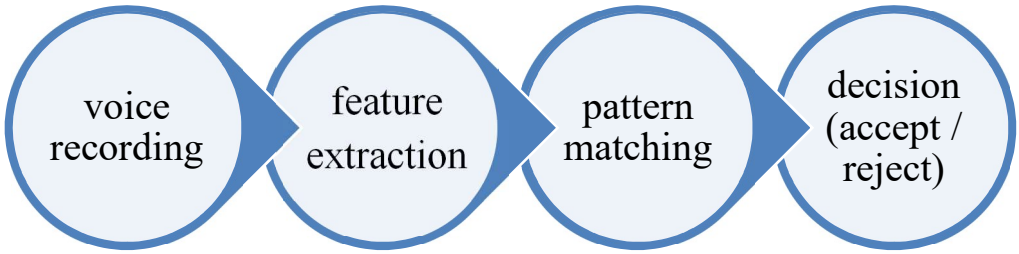
In order to repeatedly apply this technique until we are left with two-point DFTs, we have to choose the length of the original DFT to be a power of two, i.e., N=2ν. In this case, after log2(N) iterations, we will reach two-point DFTs. As Figure 3 suggests each stage of the algorithm requires N complex multiplications and hence, the total number of complex multiplications will be Nlog2(N). In the example of the eight-point DFT, we will have 8log2(8)=24 complex multiplications. Then, while the direct computation of the eight-point DFT requires 256 real multiplications, the decimation-in-time method requires only 24×4=96 real multiplications. The reader can easily verify that the computational saving will be really dramatic for large DFTs.

Recognition Criteria

There are two types of speaker recognition:

- Text dependent (restrained) : the subject has to say a fixed phrase (password) which is the same for enrollment and for verification, or the subject is prompted by the system to repeat a randomly generated phrase.
- Text independent (unrestrained) : recognition based on whatever words the subject says.

Basically identification or authentication using speaker recognition consists of four steps which are presented below:



Depending on the application, a voice recording is performed using a local, dedicated system or remotely (e.g. phone). The accoustic patterns of speech can be visualized as loudness or frequency vs. time. Speaker recognition systems analyze the frequency as well as attributes such as dynamics, pitch, duration and loudness of the signal.

During feature extraction the voice recording is cut into windows of equal length. After that, we move on to pattern matching which is the actual comparison of the extracted frames with known speaker models; this results in a matching score which quantifies the similarity in between the voice recording and a known speaker model.

In order to obtain the best results, we had to consider several recording and speaker recognition criteria. The first criterion that we took into consideration was choosing the right method of recording. There are no particular constraints on models or manufacturers when using regular PC microphones, headsets or the built-in microphones in laptops, smartphones and tablets. However these factors should be taken into consideration:

- The same microphone model is recommended (if possible) for use during both enrollment and recognition, as different models may produce different sound quality. Some models may also introduce specific noise or distortion into the audio, or may include certain hardware sound processing, which will not be present when using a different model. This is also the recommended procedure when using smartphones or tablets, as different device models may alter the recording of the voice in different ways.
- The same microphone position and distance is recommended during enrollment and recognition. Headsets provide optimal distance between user and microphone; this distance is recommended when non-headset microphones are used.

The next criterion we will be talking about refers to sensitivity. Speaker recognition programs are usually sensitive to noise or loud voices in the background; they may interfere with the user's voice and affect the recognition results. We found several solutions that may be considered to reduce or eliminate these potential problems:

- A quiet environment for enrollment and recognition.
- Several samples of the same phrase recorded in different environments can be stored in a biometric template. Later the user will be matched against these samples with much higher recognition quality.
- Close-range microphones (like those in headsets or smartphones) that are not affected by distant sources of sound.

The last criterion that we took into account was the change that can appear in the natural voice. The following changes may affect speaker recognition accuracy:

- a temporarily hoarse voice caused by a cold or other sickness;
- different emotional states that affect voice (i.e. a cheerful voice versus a tired voice);
- different pronunciation speeds during enrollment and identification.

Characteristics

The performance of a speaker recognition system is characterized by two essential factors: complexity and robustness.

In general terms, the complexity of a system refers to the processing complexity and the complexity of the projected model. The processing complexity includes the cost of the processing time of each subcomponent of the model. In the case of real speaker recognition systems, where the execution of operations must be done in an efficient time, the processing complexity is of crucial importance.

At present, several speaker recognition systems are trained on a set of data collected in certain situations. These systems would work correctly if the processing conditions were equivalent to the situations in which the data were collected. Unfortunately, most of the time, these conditions are not met because these differences are inevitable. Essential aspects regarding the processing conditions of these systems include background noise level, communication channel distortion, communication channel noise, difference in speech styles, syntactic deviation, etc. In practice, the deviation of these conditions from the features established during the design phase of the system can substantially affect its performance. Thus, robustness is an essential concern, becoming a critical performance indicator of all speaker recognition systems.

Our project intended to recognize who is speaking by comparing the newly recorded voice with the already existent voices from the database, which makes it text independent. The main criterion that led us to achieve favorable outcomes is the uniqueness of the voice pitch. The pitch value is specific to each individual. This value corresponds to the natural tone of our voice. Men have a pitch value located somewhere around 120 Hz, and women around 220 Hz. From its average value, the pitch varies during an utterance as people can change their voice from deep to high-pitched. Typically, the pitch value of a man can range from 50 Hz to 250 Hz, and for a woman, from 120 Hz to 480 Hz. Furthermore, we tried to record using different devices: phones, laptops and a singing microphone so we could check the performance and stability of our program. Robustness is very dependent on the setup, so when phones or computer microphones are used the algorithms will have to compensate for noise and issues with room acoustics. Next, since we could not record our samples in a noise-free environment, we tried to reduce the amount of outside noise by recording in our rooms at home and in an empty classroom provided by our University. Again, we tried both these options because we wanted to see how accurate our program is in both situations. Last but not least, we made sure no one was sick or had fluctuations in his/hers emotional state so our program could work at its best without being misled by the voice changes that may appear during these conditions. Moreover, everyone recorded at the same pace, so there were no differences regarding pronunciation speeds. These are the main factors we took into account when we started working on our project.

The results we achieved after running the program for the first time were decent. The main drawback that we encountered was the time. Our software took a long time to display the desired results, used a lot of memory and the CPU usage was fairly high.

For this reason, we started to look at the performance status as well. In this regard, we tried to optimize our C++ classes so that they were less greedy when it came to CPU and RAM usage. In the following pages presenting the case study, the description of our program as well as the optimization process will be put forward. Also, we are going to present the final results, which were reached faster and by using fewer resources.

Case Study

In this paper, a C++ speaker recognition software is presented. In order to achieve this result, our team, consisting of 7 individuals, started by studying the documentation provided by our guiding teacher as well as other well-known papers on this subject. This first step took us some time because our initial knowledge on this matter was very poor. To speed things up, we dissociated the problem in:

- Voice recording and WAV converting
- Sampling frequency application
- Fast Fourier Transform
- Comparison
- Results

Then we formed mini-groups of 1 or 2 people, each group studying a certain aspect of the problem.

The next step was combining all that we had learned and come up with solutions to the problem, which made our final product look like this:

1. Our project started with recording our voices, converting them to *WAV* files and adding them to a database.
2. We selected from each of those recordings approximately 10 seconds so as we could find both the biggest and the lowest voice spike. We also tried to keep the RAM and CPU usage at their lowest and get the best results in the smallest amount of time. After that, the file was converted so it was expressed in 32 bits (16 bits would have been harder to work with since *float* values in C++ are expressed in 32 bits).
3. In the following bytes, what was important for us was the amplitude. Since we had stereo recordings, we had to separate all the info stored there in two different channels: left channel and right channel. Each amplitude was then saved in 32 bits like this: the first amplitude was written in a vector (left channel), the following one was written in another vector (right channel), the next one was written in the first vector, and so on. Since the left and the right channels had very similar information, we chose to use only the vector which stored the info from the right channel in order to make the program run as fast as possible and at its best performance without using too many resources. Then we applied Fast Fourier Transform (FFT) on that vector, so the initial vector was transformed into a vector of complex numbers. We then saved the absolute values of those complex numbers in the same vector by overwriting them.
4. We normalized the vector. Then, we noticed that our sampling frequency (a *float* number expressed in 8 bits) was around 44100 for each of our recordings.
5. We divided this frequency to the length of the vector discussed above and put the first half of these numbers in another vector due to the same reason as before, so we had a corresponding amplitude for each frequency.

6. Passing on to the comparison, we took the highest value from the amplitude vector for both the samples we recorded and the test voice and memorized the index adequate to the maximum value of the amplitude, searched for the corresponding frequency and saved the fitting indexes in a vector.

7. For each sample, we searched for the indexes corresponding to the frequency found before in the test voice.

8. For each sample, we used those indexes to search through the amplitude vector and save the highest one. We took the highest value using a margin of error of ±10 indexes and checked which recorded sample it corresponds to.

9. Last, we simply displayed the name of the person whose voice resulted from the test. In order to do so we created 3 different intervals: the first one (>80%) shows that the test voice is very similar to one of the voices in the sample database, so that person's name is displayed on the screen; the second one (80%-50%) means that it is highly probable that the voice used for the test can be found in the database and that person's name is shown ; the third one (50%-20%) shows that there are small chances for that voice to be found in the database, but nonetheless the name of the person in the database whose voice resembles the best the one we used for the test is displayed. If the percentage is below 20%, we considered that the voice was not found in the already existent database.

At first, we found it simple to use MATLAB, which is known for being an easy environment for signal manipulation, for implementing our program, since the Fast Fourier Transform was easier to apply there and get the expected results. Then we tried to export it to C++, but that did not work the way we wanted so we were back to square one. We chose to stick to the C++ implementation but since we were new to the subject of speaker recognition, we felt the need to check the accuracy of our results by comparing them with the results exported from MATLAB.

Dividing our program in several classes resulted in a noticeable improvement in time and memory usage. In the beginning, it took our software 56 seconds to display the first results. After the fragmentation, the time needed decreased to only 8 seconds, so our performance was improved by 85.6%. However, our program's chance of succes is around 80%. It encounters difficulties when it has to compare two female or two male voices whose frequencies are similar. We plan to take care of this inconvenience in our future research.

Next, we will present the most important fragments of our C++ program, which are the Fourier Class and the Comparison Function.

As we explained earlier, the Fourier Class is dividing our wave input into sinusoids (sine waves) of different frequencies and stores the amplitude and the corresponding frequency in two different vectors. A fragment of the code is displayed below:

```
199  void Fft(CArray& x)
200  {
201      const size_t N = x.size();
202      if (N <= 1) return;
203
204      // divide
205      CArray even = x[std::slice(0, N/2, 2)];
206      CArray  odd = x[std::slice(1, N/2, 2)];
207
208      // conquer
209      Fft(even);
210      Fft(odd);
211
212      // combine
213      for (size_t k = 0; k < N/2; ++k)
214      {
215          Complex t = std::polar(1.0, -2 * PI * k / N) * odd[k];
216          x[k] = even[k] + t;
217          x[k+N/2] = even[k] - t;
218      }
219  }
```

Figure 1 - Fourier Class Preview

Our Comparison Function searches for and stores the maximum value of the amplitude vector created using FFT and retrieves its adequate index. Then it verifies in the frequency vector the value corresponding to the specified index mentioned before. The main code is the following:

```
381
382  }while((inputBaza <=9 && inputBaza >=1) && (counterBaza <dimenisuneBazaVerificare));
383
384  valoareFinalaMaxima = -1;
385  for(i=0; i<dimenisuneBazaVerificare; i++)
386  {
387      if (freArray[bazaVerificare[i]] > valoareFinalaMaxima)
388      {
389          valoareFinalaMaxima = freArray[bazaVerificare[i]];
390          valoareIndice = bazaVerificare[i];
391      }
392  }
393
```

Figure 2 - Comparison Function Preview

Diagrams

We chose to use Unified Modeling Language 2 in order to make our program easier to understand. UML2 has many types of diagrams, which are divided into two categories. Some types represent structural information, and the rest represent general types of behavior, including a few that represent different aspects of interactions. Below are several such diagrams which helped us structure our program. They present the connection the user has with the software as well as the relationships between the classes, methods and objects used for building this project.

- An **use case diagram** is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.

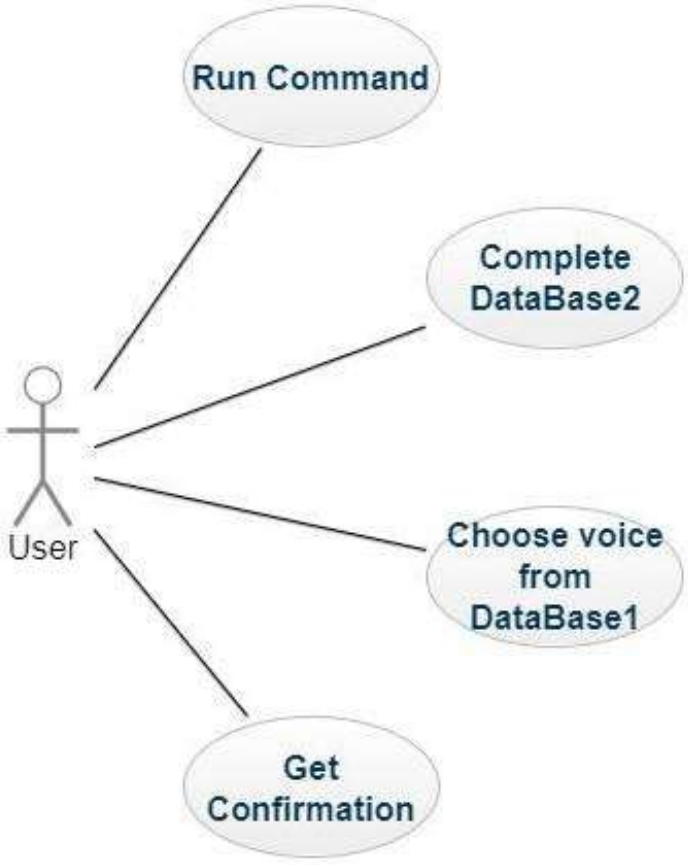


Figure 3 – Use Case Diagram

This is the simplest of the diagrams presented in this paper and it illustrates the base commands the user has access to. First, the user starts the program by pressing the „Run Command" button. Then he has to complete DataBase2 and choose the desired voices which will be used for comparison from DataBase1. Finally, after the analysis is over, he will get the confirmation that

the voice selected from DataBase1 is similar to the one chosen from DataBase2 or that the voices do not match.

- A **class diagram** in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

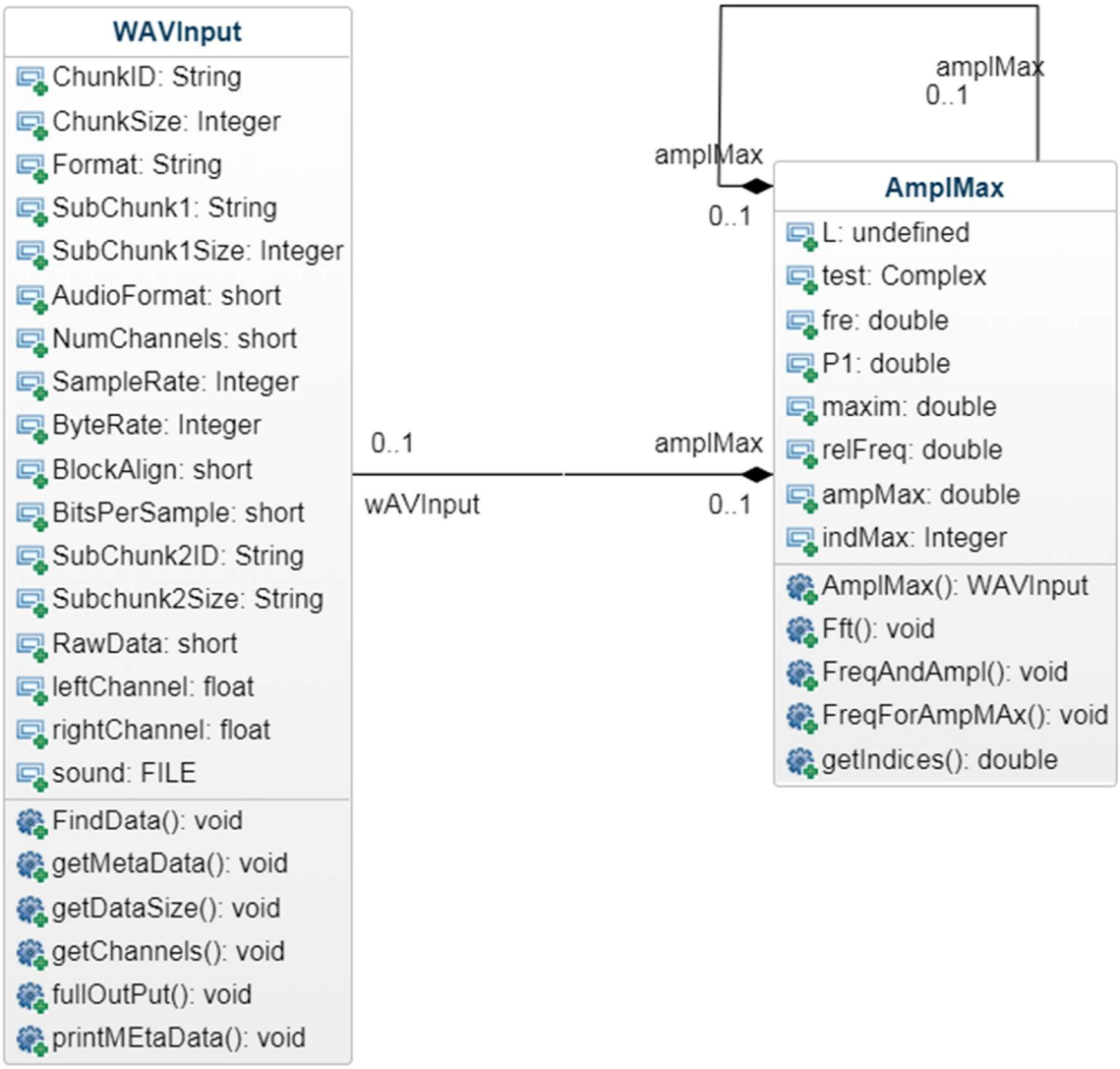


Figure 4 - Class Diagram

This a little bit more complicated diagram presents the two classes that have been built in C++ (WAVInput and AmplMAX) with their particular methods and objects. WAVInput class has been broadly explained before in the case study. FreqAndAmpl method from AmplMAX class creates the amplitude-frequency vectors, beginning with the frequency of 0 and ending with the maximum frequency found (basically, every frequency is assigned a corresponding amplitude).

The FreqForAmpMax method finds the frequency which corresponds to the maximum amplitude from the test voice. The getIndices method gets the amplitude (from every voice in the database) which meets the maximum frequency found for the test voice. Also, the relationship between these two classes is shown in this diagram, which is composition. Moreover, since class AmplMAX is recurssive, it is in a composition relationship with itself.

- The **activity diagram** is another important diagram in UML describing the dynamic aspects of the system. It is basically a flowchart to represent the flow from one activity to another activity.

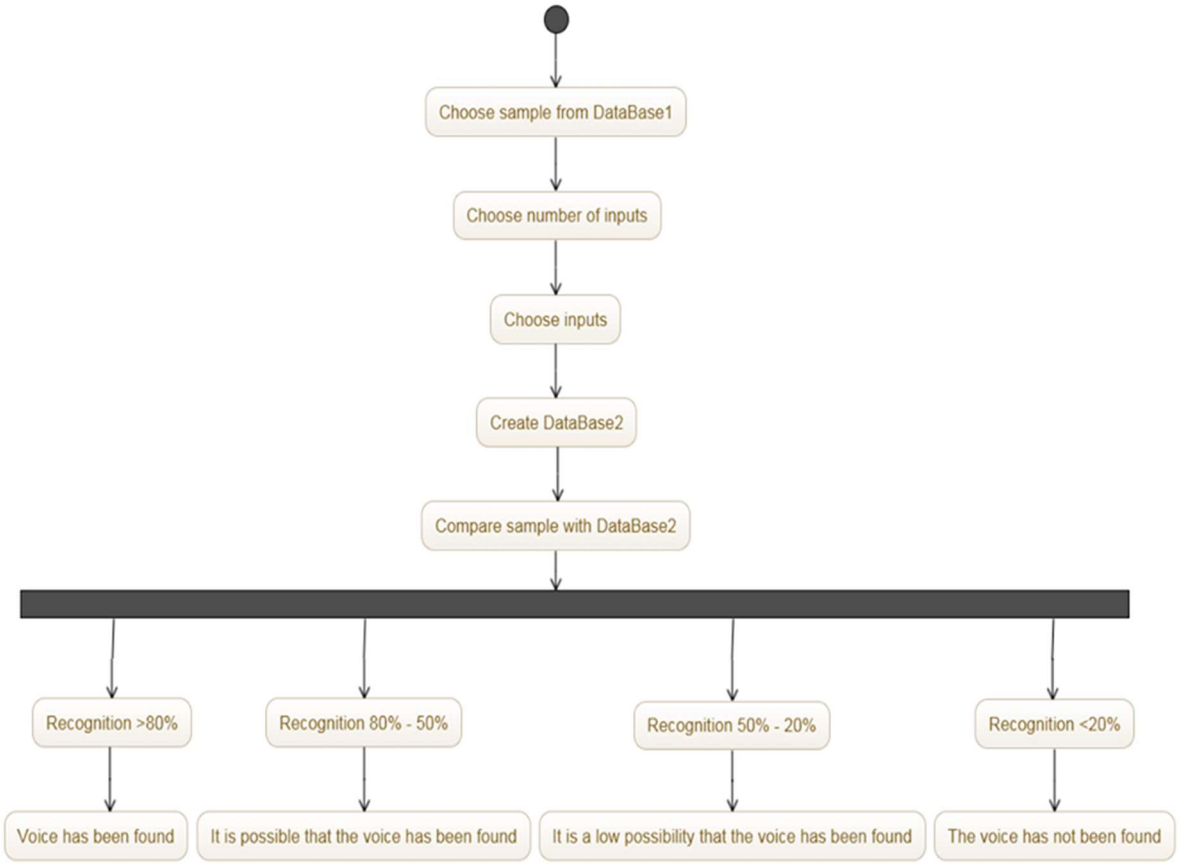


Figure 5 - Activity Diagram

The Activity Diagram presents the steps which lead to getting a result from analysing the voice samples. Firstly, it is needed to pick a voice sample from DataBase1. Secondly, the user has to choose the number of voice samples with which he wants to compare the sample chosen in the previous step. Thirdly, he has to effectively pick the samples which will be used to fill DataBase2. Then, the actual comparison will take place. Last, the result will appear on the screen, showing different percentages depending on the degree of similarity between the samples chosen from the two databases.

- The sequence diagram is used primarily to show the interactions between objects in the sequential order that those interactions occur. They are useful in documenting how a future system should behave.

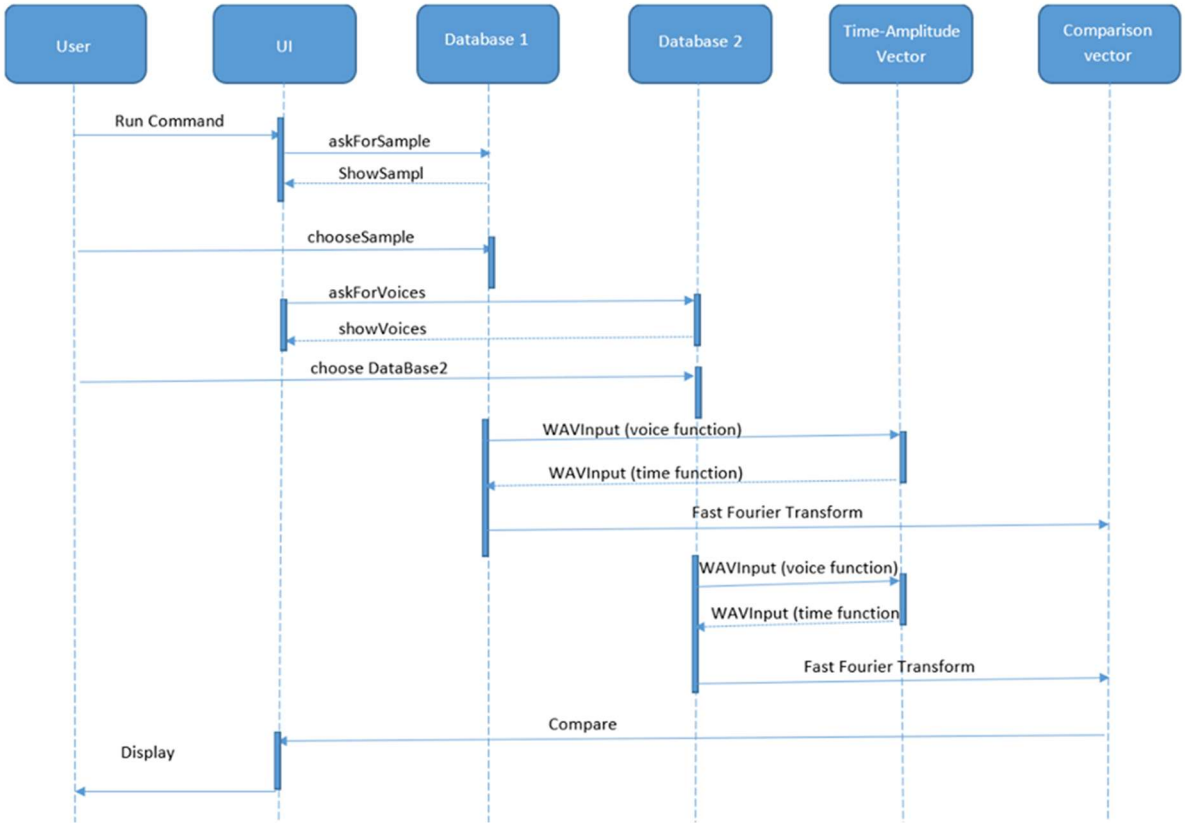


Figure 6 - Sequence Diagram

This last diagram presents how the program works in the background. The interractions between the classes and the objects are also shown. To cut long story short, the user runs the command and chooses the sample and the voices he wants to use. Then the WAVInput class turns them into amplitude vectors which are next transformed using FFT. Last, the product resulted from the comparison function is displayed.

Conclusions

The objective of this project was to design and implement a system that could identify the speaker by comparing his/her voice with the voices already existent in the pre-recorded samples. The analysis presented in the previous sections shows a fully functional speaker recognition program, in which Fast Fourier Transform along with a frequency comparison function was used.

Overall this project could have been a great success. However, due to time constraints and the complexity of what the team wanted to do impeded the integration of all optimization modules and the completion of our project. In the future work, we will try to optimize our software by coming up with more efficient algorithms and find a method to compress the data extracted from the recordings, and try to improve the efficiency of the proposed system by making it less sensitive to the unwanted but innevitable outside noise.

Some of the lessons learned from this project are the importance of a good documentation, a good design planning and long hours of testing and debugging, as well as a good cooperation and coordination between the members of the group. As a whole, this project gave insight into building a complex system which is anchored in the reality of 2018. Password typing is long gone and it is the time for human voice to shine and create a new world of possibilities by facilitating our safe existence.

Acknowledgments

We are thankful to all researchers and authors of various manuscripts from where we got valuable information which helped us complete the work. Also, it is needless to say that accomplishing this project was possible thanks to our Object Oriented Programming teacher, M. Caramihai.

References

Shoup, A., Talkar, T., Chen, J. and Jain, A. (1994). An Overview and Analysis of Voice Authentication Methods.

Saady, M. R., El-Borey, H., El-Dahshan, E. A. and Yahia, A. S. (2014). Stand-Alone Intelligent Voice Recognition System. Journal of Signal and Information Processing.

Diaz, J. and Muniz, R. (2007). Voice Recognition System.

Bellanger,M. (2006). Traitement numérique du signal : Théorie et pratique. Dunod, Paris, 8e edition.

Sharma, V. and Bansal, P. K. (2013). A Review On Speaker Recognition Approaches And Challenges. International Journal of Engineering Research & Technology (IJERT).

Iyer, S. C. (2014). Speaker Recognition System using Coefficients and Correlation Approaches in MATLAB. International Journal of Engineering Research & Technology (IJERT).

Smith, S. W. The Scientist and Engineer's Guide to Digital Signal Processing.

Arar, S. (2017). An Introduction to the Fast Fourier Transform [https://www.allaboutcircuits.com/technical-articles/an-introduction-to-the-fast-fourier-transform/].

Fast Fourier Transform. [http://www.cmlab.csie.ntu.edu.tw/cml/dsp/training/coding/transform/fft.html].

MathWorks. [https://www.mathworks.com/help/matlab/ref/fft.html?requestedDomain=true].

Biometric Solutions 2016. [http://www.biometric-solutions.com/speaker-recognition.html].

WAVE PCM soundfile format. [http://soundfile.sapp.org/doc/WaveFormat/].

GenMyModel. [https://api.genmymodel.com/].