# Automatic Accompaniment Music Generation and Singing Skill Evaluation for Vocal Melodies

K.A.K.Indrachapa
Index No : 13000454

D.H.U.Perera
Index No : 13000901

R.W.M.N.H.Wanigasekera
Index No : 13001264

W.K.P.Wanniachchi
Index No : 13001272

Supervised by

Dr. K. L. Jayaratne

Submitted in partial fulfillment of the requirements of the
B.Sc. in Software Engineering (Hons) Final Year Project in Software Engineering
(SCS4123)


University of Colombo School of Computing
Sri Lanka
2017

# Declaration

We certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of our knowledge and belief, it does not contain any material previously published or written by another person or ourselves except where due reference is made in the text. We also hereby give consent for our dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name : Ms. K.A.K. Indrachapa

Signature of Candidate : ..... Date : .....

Candidate Name : Mr. D.H.U. Perera

Signature of Candidate : ..... Date : .....

Candidate Name : Mr. R.W.M.N.H. Wanigasekera

Signature of Candidate : ..... Date : .....

Candidate Name : Mr. W.K.P. Wanniachchi

Signature of Candidate : ..... Date : .....

This is to certify that this dissertation is based on the work of
Ms. K.A.K. Indrachapa, Mr. D.H.U. Perera, Mr. R.W.M.N.H. Wanigasekera, Mr. W.K.P. Wanniachchi under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Supervisor: Dr. K.L. Jayaratna

Signature of Supervisor : ..... Date : .....

# Abstract

Music composition may be one of the most difficult tasks for non-musicians. To allow non-musicians to get a taste of music creation, we propose a system for generating accompaniment music for vocal melodies.

In the system that we propose, a pitch detection module first extracts the pitch classes from the vocal melody using a time domain auto-correlation method. According to these pitch classes, a singing skill evaluation module classifies the skill of the vocal melody as good or poor based on the pitch and the tempo. The pitch based singing skill classifier uses two features of pitch interval accuracy and classifies the singing skill using a trained Support Vector Machine. The tempo based singing skill classifier uses a vibrato suppression based onset detection technique to estimate the tempo of the vocal melody and classifies the singing skill. A Hidden Markov Model with learned probabilities then construct the best acceptable chord progression for the vocal melody by applying the Viterbi algorithm. The system then generates the accompaniment music using the constructed chord progression and combines it with the vocal melody to obtain the final outcome.

Finally, we present the results from the first and the second study demonstrating that the performance of singing skill classifiers is fairly good with an accuracy of 81.8% and 87.5% respectively. We present results from the third study showing that our system is able to construct acceptable chord progressions for vocal melodies at a rate of 89.381% accuracy. The results from the final study shows that the users are enthusiastic about the final outcome of our system with above 3.76 mean score for all the Likert-scale questions.

# Acknowledgement

We would like to express our sincere gratitude to our supervisor, Dr. K.L. Jayaratne, senior lecturer of University of Colombo School of Computing for providing us continuous guidance and supervision throughout the research.

We would also like to extend our sincere gratitude to Dr. A.R. Weerasinghe, senior lecturer of University of Colombo School of Computing and Mr. G.K.A. Dias, senior lecturer of University of Colombo School of Computing for providing feedback on our project proposal and interim evaluation to improve our study. We also take the opportunity to acknowledge the assistance provided by Dr. M.I.E. Wickramasinghe as the coordinator of the final year software engineering project.

We would also like to acknowledge Mr Suresh Maliyadde, outstanding music director and one of the most experienced musicians in the current Srilankan field of music, for the volunteered service by allocating his valuable time to advice and feedback us in designing our project. We would also like to thank all the musicians for being generous to support expert evaluation processes.

Our deepest gratitude goes to our loving families for their unconditional support, love and encouragement extended towards us throughout this journey of life. Finally, it is a great pleasure for us to acknowledge the assistance and contribution of all the people who helped us to successfully complete our project.

# Contents

# List of Figures

# List of Tables

# Acronyms

| | |
|---|---|
| AIFF | Audio Interchange File Format |
| API | Application Program Interface |
| DFT | Discrete Fourier Transformation |
| FLAC | Free Lossless Audio Codec |
| HMM | Hidden Markov Model |
| HTTP | Hypertext Transfer Protocol |
| IEC | International Electrotechnical Commission |
| IEEE | Institute of Electrical and Electronics Engineers |
| ISO | International Standards Organization |
| MAIE | Mean Absolute Interval Error |
| MAPE | Mean Absolute Pitch Error |
| MFCC | Mel Frequency Cepstral Coefficient |
| MIDI | Musical Instrument Digital Interface |
| MMA | Music Midi Accompaniment |
| MT | Machine Translation |
| REST | Representational State Transfer |
| SOAP | Simple Object Access Protocol |
| STFT | Short Term Fourier Transform |
| SVM | Support Vector Machine |
| UI | User Interface |
| WER | Word Error Rate |
| XML | Extensible Markup Language |

# Chapter 1

# Introduction

## 1.1 Background

A song is usually defined as a combination of vocal melody and instrumental accompaniment. In the process of professional singing, a musically satisfying arrangement of notes is performed by a singer with rhythmic and harmonic support provided using musical instruments. The musically satisfying arrangement of notes is referred to as the melody whereas the support provided by the musical instruments is generally referred to as the accompaniment music. Figure 1.1 illustrates the basic structure of a song.

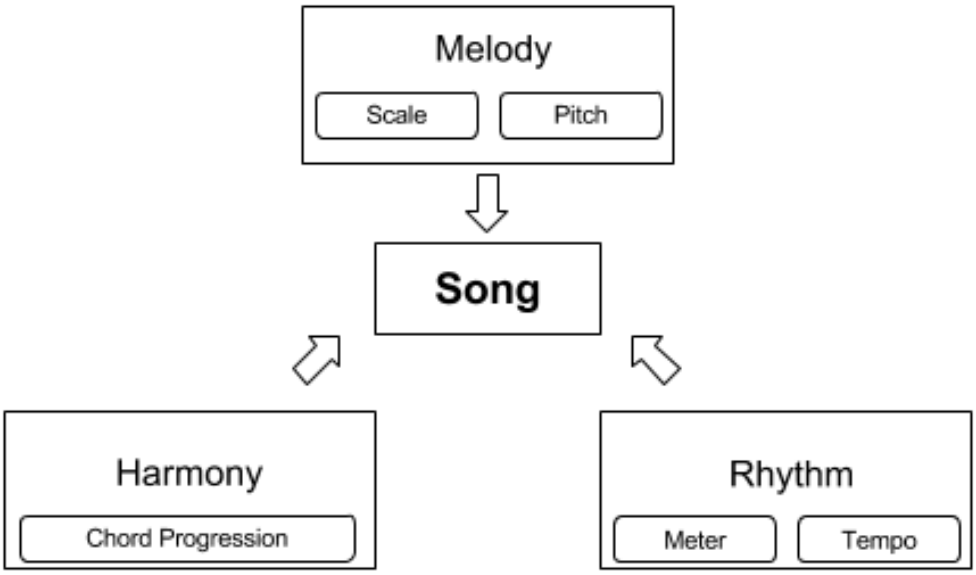



Figure 1.1: Basic structure of a song

A melody is a sequence of notes arranged in a musically satisfying manner while a note is a sound which consists of a particular pitch and a duration. When there is more than one pitch sounding at the same time in music it results a harmony. A chord is a different kind of harmony which have three or more notes sounding simultaneously. An arrangement of chords is referred to as a chord progression and there can be one or more acceptable chord progressions for a single melody. Chord progressions are usually played using musical instruments such as piano, guitar etc. to accompany a melody. The final element of a song is the rhythm which defines the repetitive pulse of the music. The speed of the rhythm is referred to as tempo and it is usually measured using beats per second.

## 1.2 Problem Definition

Knowledge of music is often required for composition of songs since the normal procedure to follow is to begin with an idea for the melody and then developing chords and accompaniment patterns to turn that melody into a song. However, individuals who lack musical knowledge would have to struggle when developing and experimenting their musical ideas. This solution will cater their problem by taking the burden of generating the chords and the accompaniment.

## 1.3 Project Aim and Objectives

The ultimate aim of this project is to allow non-musicians to get a taste of music composition using a mobile application. This application is capable of generating accompaniment music when a user sings a song. Furthermore, providing such a service would create an entertainment value.

There are two key objectives in order to achieve the project aim. The first objective is **identifying the singing skill of the user**. The acceptability of generated accompaniment music depends on the singing skill of the user. Encouraging the user to improve their singing skill would increase the acceptability of generated accompaniment music. The second objective is **selecting an acceptable chord sequence for a vocal melody**. A core requirement of an accompaniment music is the chords progression.

Even though this project is aimed at delivering a feature-rich accompaniment music generation application for entertainment purposes, it will have the capability of delivering basic functionalities for advanced users who compose music commercially in the future enhancements of the product.

## 1.4 Delimitation of Scope

It is beyond the scope of this study to examine the internals of accompaniment generation software. This study aims to contribute on automatic harmonization of vocal melodies so that the produced chord progression can be used as an input to the accompaniment generation software to get the final accompaniment music.

## 1.5 Dissertation Outline

This dissertation has been organized in the following way. Chapter 2 lays out the background study, related work and similar systems whereas the Chapter 3 describes functional and non-functional requirements of the project. Chapter 4 begins with the overall design and further explains the architectural design and the software development life cycle of the project. The design of singing skill evaluation under pitch based and tempo based evaluation is described in Chapter 5. Chapter 6 presents the design of harmonization of vocal melodies using an HMM. Chapter 7 begins by laying out the tools and technologies used and looks at the implementation of the server and the mobile application. Chapter 8 presents the results and evaluation of the project and finally, Chapter 9 concludes with a summarization of the work carried out and suggested future work.

# Chapter 2

# Background Study

This chapter discusses approaches and systems related to this project. Section 2.1 elaborates more on related work and similar systems. Section 2.2 reviews recent research approaches on automatic harmonization of vocal melodies and Section 2.3 presents research conducted on singing skill evaluation under pitch based and tempo based singing skill.

## 2.1 Related Work

A typical approach to allow users to get a taste of music creation has been to use music production software that has pre-programmed musical structures such as rhythm patterns, music clips etc. The users would record, experiment and combine the musical structures in order to come up with a final music creation. However, musically untrained users might find it difficult to use this approach due to lack of domain knowledge in music.

A more suitable approach would be to provide users with an accompaniment music track to sing along with it. The accompaniment music track can be a pre-produced track based on a popular song. This approach is identified as Karaoke[1] and Sing!Karaoke by Smule[2] is an implementation of Karaoke through a mobile application. Despite its quality and rich music tracks, Karaoke suffers from several major drawbacks. A musically untrained user may only be able to engage with a limited variety of songs and that user would not be able to create new music.

A solution for the aforementioned drawback would be to generate the accompaniment music track automatically according to the vocal melody which the user sings. A system which follows this approach is Microsoft SongSmith.

[1] A form of interactive entertainment or video game developed in Japan in which an amateur singer sings along with recorded music using a microphone. The music is typically an instrumental version of a well-known popular song.

[2] https://www.smule.com

### 2.1.1 Microsoft SongSmith (MySong)

Microsoft SongSmith (previously known as MySong) is a standalone system which generates accompaniment music automatically for a given vocal melody. The core of automatic accompaniment generation is based on the extraction of chords from a vocal melody. SongSmith software application uses a Hidden Markov Model which was trained using a lead sheet database to select chords for vocal melodies[1]. The model parameters are intuitively exposed to the user so that the user can experiment with the generated chord sequence[2]. A major drawback of 'SongSmith' is that the system tends to generate accompaniment which is out of tune with the melody input if the user does not sing in a standard musical chromatic key. As a solution to this problem, a singing skill evaluation method was introduced in this project.

## 2.2 Automatic Harmonization of Vocal Melodies

A chord progression is an important component in an accompaniment music generation system. Given a chord progression according to a melody, an accompaniment music generation software can produce music using various rhythm patterns according to a predefined tempo. To generate accompaniment for any given melody, the problem of automatic harmonization should be addressed.

Several studies investigating automatic harmonization used pitch class profile to come up with chord progressions. Fujishima has suggested a realtime software system which recognize chords from a polyphonic audio signal [3]. The pitch class profile of the audio signal was derived from the Discrete Fourier Transformation(DFT) spectrum of the audio and a pattern matching technique was applied to obtain the chord type. However, one major limitation of pitch class profile based chord recognition is that they require a polyphonic audio signal. Chuan and Chew presented a hybrid system for generation of style-specific accompaniment [4]. Neo-Riemannian transforms was used to construct the chord progression list from MIDI melodies and the final chord progression was generated using a Markov chain with learnt probabilities for the transforms. But one major limitation of this kind of systems is that they require users to input the melody in formal musical formats like MIDI or score. The most promising method of automatic accompaniment for vocal melodies was suggested by SongSmith [1].

## 2.3 Singing Skill Evaluation

Several attempts have been made to evaluate the singing skill using various approaches. Previous research findings show that pitch (drift, intonation, vibrato) and tempo (onset detection) can be used as factors to measure singing skill evaluation[5].

Different approaches of singing skill evaluation based on pitch related measures (vibrato, pitch interval accuracy, drift, intonation) are described below.

### 2.3.1 Pitch Based Singing Skill Evaluation

Mauch et al. present a way to measure intonation (accuracy of pitch in playing or singing) in unaccompanied singing[6]. Definition of intonation brings 'reference pitch' into play which can be either external or internal. They have introduced two metrics to measure accuracy and drift which are interval error and pitch error.

An interval is known as the distance between two pitches in music and given by equation 2.1.

$$\Delta p_i = p_i - p_{i-1} \tag{2.1}$$

Where $p_i$ is the $i^{th}$ pitch and $p_{i-1}$ is the pitch of the preceding note. $\Delta p$ gives the signed distance between the $i^{th}$ and $(i-1)^{th}$ pitch. Therefore interval error of an observed interval is denoted by [6, eq. (2)]. Second measure is introduced as pitch error. However since there's no external reference pitch in unaccompanied singing pitch error has not been defined directly. Therefore an estimate for the tonic[3] pitch is defined in [6, eq. (3)] Using those metrics singing accuracy and precision metrics are explained as mean absolute pitch error (MAPE) and mean absolute interval error (MAIE) in [6, eq. (5)]and [6, eq. (6)] respectively.

Major concern of this approach is that it requires the score information of the song to compute the nominal interval and pitches. Since we are evaluating the singing skill of unknown melodies score information is not available.

ZWAN presented an artificial neural network based method for automatic recognition of quality of singing[7]. According to the study they have defined the quality factor as a subjective measure of whether the voice belongs to an amateur or to a professional singer. They have used 331 parameters which consists of glottal source, vibrato and intonation parameters. Glottal source parameters are defined in the time domain and computationally expensive. A person who maintains a stable vibrato can be identified as a professional singer. Therefore vibrato was used as another parameter. Vibrato parameters were extracted using pitch contours. They have used a feed forward neural network for the singing voice quality recognition and the results obtained were compared with the expert judgments. The aim of this study [7] is to distinguish the performance between an amateur and a professional singer whereas our requirement is to evaluate the singing skill of musically untrained users as good or poor. But we considered the overall approach where classification was done using a neural network and kept looking for more relevant features.

[3]Tonic-also known as the keynote. Tonic is the first note of any scale.

Ex: if key is C major then tonic is C.

Another approach was suggested for singing skill evaluation of unknown melodies by Nakano and Goto [8]. Classification using an SVM (support vector machine), pitch interval accuracy and vibrato features were suggested. Pitch interval accuracy and vibrato are known to be independent from singer and the melody. Therefore these features were used to evaluate the singing skill where score information is unavailable. Pitch interval accuracy is defined by how well a fundamental frequency F0 trajectory fits in to a 100 cents width grid. Semitone stability $P_g(f,t)$ was then defined using pitch interval accuracy as given in [8, eq. (3)] and shows the semitone stability in good and poor singing respectively [8, Fig.2 ]. Vibrato is the other feature they have considered. It is known as the periodic fluctuation of F0. The study has proposed a method to estimate vibrato sections using Short Term Fourier Transform (STFT). Vibrato likeliness was used as a metric to decide whether an estimated section is an actual vibrato section [8, Fig.4]. As the next step of the study they trained an SVM using the features extracted from semitone stability (related to pitch interval accuracy) and vibrato sections.

We used this approach suggested by Nakano and Goto because it is compatible with our requirements but we only considered the pitch interval accuracy related features as vibrato is an advanced technique in singing. Vibrato is mostly used by musically trained users and professional singers[4], but our requirement focuses more in evaluating the singing skill of musically untrained users. Therefore vibrato was not taken into consideration.

### 2.3.2 Tempo Based Singing Skill Evaluation

Much of the current literature on tempo detection of songs pays particular attention to onset detection methods. Alonso et. al present an onset detection based tempo extraction algorithm using a concept of spectral energy flux[9]. It involves three stages. They are as extraction of onset by front-end analysis, periodicity detection block and temporal estimation of beat locations.

Most of the studies use these three stages to tempo detection with different kind of onset detection mechanisms and periodicity detection techniques. According to this study, onset detection function is based on the spectral energy flux of the input audio signal.

But the generalizability of much published research on this onset detection is problematic for extracting tempo in vocal melodies because the singing voice can be seen as a pitched non-percussive instrument which does not have rhythm information[10].

Spectral onset detection can be of two types as energy based spectral onset detection methods and phase based onset detection methods. Energy based methods are useful for strong onsets, such as percussive instruments, whereas the phase based methods are good for soft onsets. Most of the voice signal contains soft onset features.

Toh et al. stated that Gaussian Mixture Models indicators have a positive impact on

[4]http://www.scielo.br/scielo.php

classifying audio features of vocal melody[11].This study aimed to introduce new onset detector based on supervised machine learning in order to capture the intricacies of singing note onsets. According to this study, the Gaussian probability distribution of the feature vector is extracted from a probability of onset and probability of non onsets. Although extensive research has been carried out on onset detection, it gives poor results for pitch based methods of unstable vocal melodies.

Bock et al. show that onset detection with vibrato suppression enhances spectral flux based onset detection[12]. According to this study, they have invented super flux algorithm by adding spectral trajectory tracking stages to common spectral flux. They have presented a detection function to pick the peaks by filtering out non onsets using median values of spectral flux.Vibrato suppression based super flux approach is chosen as a basis for onset detection because of variation of vocal melodies.

Alonso et al. stated that there is a very common problem when human taps along with music to identify tempo and automatic tempo estimation methods also make this doubling or halving of the true tempo[9]. According to this study, tempo estimation which is provided by the algorithm is labeled as correct if there is a less than 5% disagreement from the manually annotated tempo. This approach is chosen because of the disagreement adjusting method is one of the more practical ways of fixing this doubling or halving issue.

## 2.4 Summary

This chapter provided a detailed description of the background study of this project. It first laid out an overview of the similar systems, Microsoft SongSmith is one of the similar systems which generates accompaniment music for a vocal melody but it does not evaluate the singing skill. Furthermore, this chapter looked at several research approaches undertaken in automatic harmonization of vocal melodies and singing skill evaluation. A number of attempts have been made to investigate automatic harmonization of vocal melodies. Even though a considerable amount of literature has been published on automatic singing skill evaluation, most of them are based on the assumption that score information of the melody is available.

# Chapter 3

# Requirements

This chapter provides an overall description of key requirements. Section 2.1 and Sections 2.2 describe functional requirements and nonfunctional requirements respectively and focus particularly upon areas of challenge and difficulty.

## 3.1 Functional Requirements

Functional requirements address the core functionalities of this project. Requirement gathering and specification phases are addressed under this section. The user story based approach was chosen to gather requirements because it is one of the main approaches that help to find out requirements by explanation and the need of the feature. Table 3.1 describes the complete set of user stories of the system.

According to these user stories, there are three main functionalities in this system: Record a vocal melody, evaluate singing skill and generate an accompaniment music for a recorded vocal melody. There are some other sub-functionalities that requires to proceed these main functionalities. But only the key functionalities are described in this section.

### 3.1.1 Recording a Vocal Melody

A recorded vocal melody is required to generate an accompaniment music and to perform a singing skill evaluation. Therefore vocal melody recording can be identified as one of the major requirements of the system. A clear, less noisy, lossless vocal melody recording would give out good results. These conditions are the challenging areas which are needed to be addressed while designing and implementing this required feature. Only the audio recording is considered here, since video recording requires more processing power than audio recording.

### 3.1.2 Singing Skill Evaluation

The singing skill evaluation process is one of the main objectives in this project. The main purpose of singing skill evaluation is to improve the chord generation process in order to

Table 3.1: User Stories

| ID | As a | I want to | So that I can |
|---|---|---|---|
| 1 | Mobile application user | Register in the system | Use system services |
| 2 | Mobile application user | Record voice melody | Record my voice melody into the system |
| 3 | Mobile application user | Select rhythm for song | Select appropriate rhythm for song |
| 4 | Mobile application user | Select tempo for song | Select appropriate tempo for song |
| 5 | Mobile application user | Request accompaniment music | Get accompaniment music for recorded voice melody |
| 6 | Mobile application user | Check singing skill | Get evaluation about my singing skill |
| 7 | Mobile application user | Listen to the full song | Play output song |
| 8 | Mobile application user | Edit chord progression | Change accompaniment music according to my choice |
| 9 | Mobile application user | Apply voice effects | Edit recorded voice melody with audio effects |
| 10 | Mobile application user | Share the song | Share my song with friends through social networks |

provide an enhanced output. There are several kinds of singing evaluation measures in the music industry. But pitch based and tempo based evaluation are only considered in this project. Because the pitch is one of the major attributes of musical tone and tempo is the number of beats per minute which required to maintain synchronization between an accompaniment music and a recorded vocal melody. This evaluation result provides a proof of the quality of output result along with musical information. Pitch extraction of unstable notes and tempo extraction of vocal melody are challenges that need to be addressed.

### 3.1.3 Generate Accompaniment Music for a Vocal Melody

An accompaniment music generation process is the other objective in this project. Using the result of this component, a chord sequence for a particular recorded vocal melody can be extracted. An accompaniment music will be generated according to chord sequence using Music MIDI Accompaniment (MMA). Only the chord extraction process is considered here. There are some preprocessing steps that require extracting a chords sequence from

vocal melody. Some of them are rhythm selection, tempo selection, removal of noises from vocal melody etc. Extracting the chords from out of rhythm songs and tracking the quickly changing chords are some challenging areas of this accompaniment music generation process.

## 3.2 Non-functional Requirements

### 3.2.1 Reliability

An Internet connection is an essential factor for this application and if there is a network failure while using this app, there can be data loss and synchronization problems. To avoid these problems all current processes will be terminated until the Internet connection is rebuilt.

### 3.2.2 Availability

The system will be available 24 hours. It will be unavailable when there are system upgrades and maintenance. The system will also be unavailable on network failures as the system is a web based one.

### 3.2.3 Security

Google authentication will be used to authenticate the user and the web services will be protected using API keys. Through this application, user's voice tracks are recorded and stored in a server. These recorded vocal melodies must be secured. Therefore, security is one of the most important factors in this application.

### 3.2.4 Maintainability

Maintainability is considered as one of the main non-functional requirements that can be used to judge the system. The system will be well modularized for ease of maintenance. All the codes will be well commented so maintenance teams can do their jobs easily.

### 3.2.5 Portability

The proposed system is a web based android solution. Therefore any mobile device with an Android operating system will run this application without any technical issues. The independent nature of the client and server applications enhances the deployment of this solution to other operating systems.

### 3.2.6 Efficiency

The proposed solution would acquire more processing power. A high processing power would be needed since there are many processes in the application which need to be run concurrently. A processor of a mobile phone will not have the capabilities for concurrent processing of these application components. As a solution to that, the components which need heavy processing power are kept in the server side and they are being accessed by the mobile application using HTTP requests.

## 3.3 Summary

This chapter presented a detailed description of the key requirements of the project. It first laid out the functional requirements of the project: Recording a vocal melody, singing skill evaluation and generating accompaniment music for a vocal melody and looked at the non-functional requirements: reliability, availability, security, maintainability, portability, and efficiency.

# Chapter 4

# Design

This Chapter indicates the design of possible solutions in accordance with the requirements specified in Chapter 3. Section 4.1 describes the main design considerations for the specified requirements and Section 4.2 elaborates more on architectural design considerations of the system.

## 4.1 Overall Design

As described in Chapter 1, the main objective of this system is to encourage people to compose their own songs without having the ability to play musical instruments. The key functional requirements to achieve this goal were recording a vocal melody, obtaining the singing skill evaluation and generating accompaniment music automatically to the recorded vocal melody.

The following overall design to cater the key requirements is comprised of six main steps: vocal melody recording, pitch detection, singing skill evaluation, chord generation, accompaniment music generation and combination. Figure 4.1 depicts the high-level design of the system.

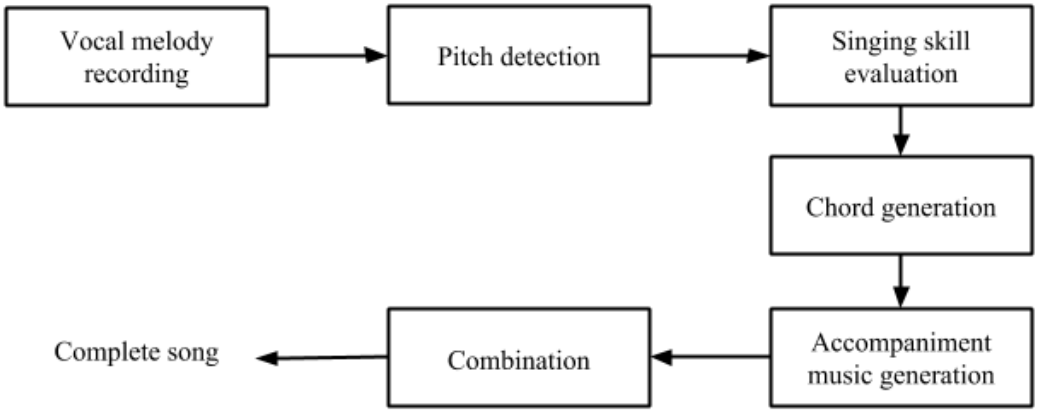



Figure 4.1: High level design of the system

### 4.1.1 Vocal Melody Recording

The vocal melody is one of the main inputs to the system and usage of a proper audio format was needed as all the following steps depend on the quality of the recorded audio. There are two different kinds of audio formats as lossless and lossy. Lossless audio formats such as WAV, FLAC, AIFF etc. preserve the original data and are larger in file size whereas lossy audio formats such as MP3, OGG etc. omit some data from the original file and are smaller in file size. Despite the large file size, lossless audio format at 44100 Hz sampling rate is used to record the vocal melody with all the relevant information that needs to detect the pitch of the recorded melody accurately.

Furthermore, the proper synchronization of the vocal melody and the generated accompaniment music requires the vocal melody to be recorded on a constant tempo since the accompaniment music is generated on a fixed tempo as specified by the user. A decision has been made to provide a rhythm track to the user while recording the melody so that the user can maintain a constant tempo throughout the recording.

### 4.1.2 Pitch Detection

Pitch detection or pitch tracking is the process of extracting the fundamental frequency from an audio signal. Time domain pitch detection and frequency domain pitch detection are the two main approaches to pitch detection. We compute the fundamental frequency using the method of Boersma [13] to compute the fundemental frequency which is a time domain pitch detection algorithm. The algorithm performs well for monophonic melodies because of its flexible, accurate and robust nature and works equally well for low pitches, middle pitches, and high pitches. The fundamental frequency of the recorded vocal melody is computed using the above mentioned algorithm at 100 sample points per second of audio and the list of generated fundamental frequency values at each point will later become an input to the pitch based singing skill evaluation and chord generation steps.

### 4.1.3 Singing Skill Evaluation

Obtaining the singing skill evaluation of the recorded vocal melody is one of the key requirements described in Chapter 3. In this study we only considered about the accuracy of the pitch and the tempo of the recorded vocal melody because of the quality of the corresponding accompaniment music will be dependent on the accuracy of the pitch and the tempo of the vocal melody. Chapter 5 elaborates more on this area and presents the overall methodology of singing skill evaluation of unknown vocal melodies. The final outcome of singing skill evaluation would be the feedback of the singing skill and the user can decide whether he/she wants to continue through the process or rerecord the vocal melody in case of a negative feedback.

### 4.1.4 Chord Generation

The core requirement of this system is to generate accompaniment music when a user sings a melody. The generation of accompaniment music depends on the chords sequence for the recorded melody. A vocal melody lack music information since it only has a monophonic audio signal. The pitch class based chord generation methods have less effect when there is monophonic audio signals. This system follows the same methodology from SongSmith which trains a Hidden Markov Model to predict the chords sequence for a given melody[1]. A detailed description of chord generation methodology is presented in Chapter 6. The output of the chord generation phase is a sequence of chords which will be an input to the accompaniment music generation phase.

### 4.1.5 Accompaniment Music Generation

Accompaniment music generation software are freely available and when provided with a chord sequence, tempo and a rhythm style, the accompaniment music can be generated. Designing for an accompaniment music generation software is out of the scope of this project.

### 4.1.6 Combination

The concluding step of the overall project design is the combination phase. The recorded vocal melody and the generated accompaniment track are combined together to generate the final outcome as a WAV file. One should not mix up the combination with a concatenation of two audio files. One on one overlap of the audio files is generally defined as the combination of two audio files. The format of the two combined files should be the same to obtain a better synchronized output.

## 4.2 Architectural Design

### 4.2.1 Hybrid Mobile Application Architecture

The current trend is using mobile devices frequently rather than using Personal Desktop computers. User interaction through a mobile device platform increases the usage of the system and encourages people to use the system more. Considering the popularity and the usability, the need of a mobile application was encountered and there were two main approaches to design a mobile application. Those were rich client mobile applications and thin client mobile applications. A rich client mobile application consists the business and the data services layers on the device itself whereas a thin client mobile application only consists the presentation and presentation logic layers on the device itself and the business and the data layers are located on a separate server.

However, the key requirement of recording a vocal melody enforced more process components such as a drum machine to play a drum sequence which has the ability to adjust the tempo live and a component to synchronize the vocal melody and the generated accompaniment music inside the mobile device. Since there were some business logic in the mobile device itself and further resource intensive business logic was available, the hybrid client mobile application architecture was followed where business and data layers resides in both device itself and a separate server. Figure 4.2 illustrates the overall system architecture.

### 4.2.2 Overview of Mobile Application Architecture

This section includes an extensive introduction to the typical layered architecture for mobile applications.

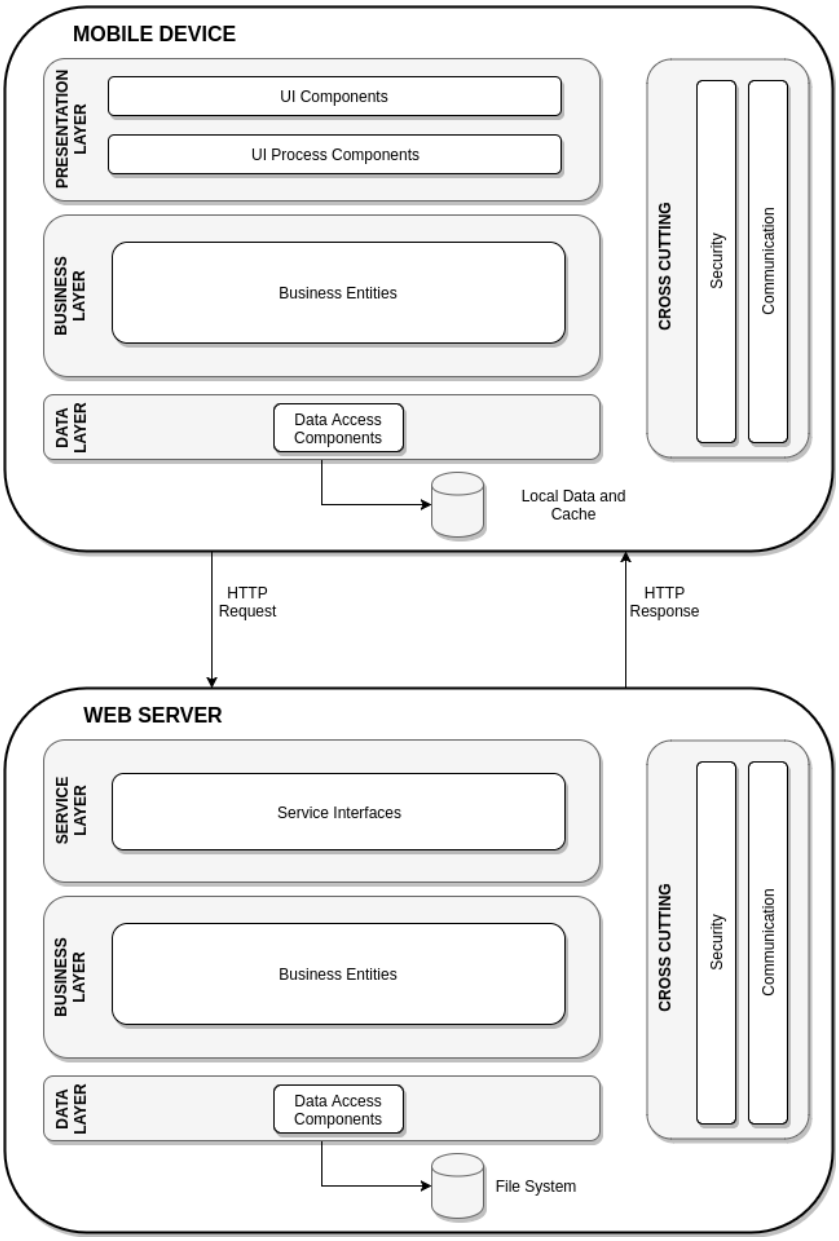



Figure 4.2: System architecture

### Presentation Layer

This layer consists of two components. UI components are the view elements for the users. UI Process components implement the user interface logic and delegates control to business logic components.

Difficulties that happens while creating an adequate mental model for the given user interfaces causes misunderstanding of using a particular mobile application. Self-explanatory and self-evident user interfaces based mechanism is identified as an effective solution for it [14]. These kinds of interfaces are introduced as inductive interfaces. There are few fundamental premises that includes to enhance these inductive interfaces: separate interfaces for separate tasks, make content to describe task and offer clear link for a task. The concept of the inductive interfaces is used in this mobile application.

To provide a build up interaction in designing the mobile application presentation layer, few aspects were considered: caching, exception management, user inputs, layouts, navigations, request processing, UI components, UI process components, and validations.

Cache only the essential data that is insensitive and non-expiring due to security and privacy manners. Design to handle all the possible exceptions and presentation of the meaningful warnings and errors were catered by the exception management unit.

Relevant input types and formats are used according to device type and information formats. Appropriate layouts are used for particular pages by user researching to achieve better effectiveness and accuracy. Manage application back stack and keep the application states for consistent navigation through pages.

Long-running requests are designed to proceed without blocking UI and appropriate processing and rendering mechanisms are used according to the requirement. Customizable UI components are created for reuse and extension purposes. Proper orchestration and synchronization mechanisms are used to design UI process component to increase the consistency and reliability of the mobile application. Critical front-end validation and constraints are added to provide a more secure mobile application.

### Business Layer

Business layer consists of the business entities which are usually known as classes. They perform the main functionalities of the application such as vocal melody recording, synchronization of vocal melody with generated accompaniment track, playing the created songs and sharing the logic of finished songs.

In the first phase of the development process, the business layer of the mobile device is considered. There, the singing skill evaluation and drum sequence generation are carried out. Due to the lack of processing power in the mobile device and to improve the efficiency of the processes the aforementioned tasks were handled in the server. In addition to above mentioned functionalities, the only task done in the mobile device is handling the requests

and responses that are made by the user and present the processed results to the user in an attractive manner.

There were few main areas that were considered in providing a better business layer while designing mobile application: business components, business entities, cohesion and coupling, concurrency and transaction control, data access, logging and instrumentation.

Business components are designed as highly cohesive components and represents a clear separation between data access and business logic. To provide programmatic access within business data, business entities are designed with relevant attributes and formats.

High cohesion and loose coupling between layers are used to improve scalability and maintainability in a mobile application. Optimistic concurrency control technique is used as concurrency and transaction control mechanisms because it provides timestamps based techniques to control concurrency without locking resources.

Appropriate data access technique formats are used in the business layer to increase the maintainability and re-usability of the mobile applications. Effective logging mechanisms are used to track and monitor system and instrumentation process is used to enhance the reliability and security aspects of mobile applications through the detection of vulnerabilities.

#### Data Layer

The data layer consists of three main components: data access logic components, data helpers, and service agents. Data access logic components provide logic to access data from a database. Data helpers provide transformation, access and manipulation activities within this layer. Service agents manage connections between exposing services with this layer.

In this mobile application, data layer consists of data access components which are used to save and retrieve mp3 formatted files from the file system and to cache the current state of the application if needed.

#### Cross Cutting

Cross cutting contains common functionalities that spans on layers and tiers. Security component of the cross cutting is responsible for authenticating the users. The communication component is responsible for establishing inter-component communication and the protocols for web server communications.

### 4.2.3 Overview of Web Server Architecture

This system can be thought of a collection of services which collaboratively work to achieve a common goal. So that Service Oriented Architecture or micro-services architecture can be used because of the separation of the tasks into services such as chord generation service, accompaniment generation service etc. However, with regard to the complexity of those

architectures and the time constraints, the layered architecture was adopted for this project because of its simplicity.

### Service Layer

Service layer works as a communicator between the server and the mobile device and all the service requests and responses from mobile device pass through the service layer. This is indeed the implementation of a RESTful API. A RESTful API is an application program interface (API) that uses HTTP requests GET, PUT, POST and DELETE to communicate with the server. In web development lot of developers generally prefer REST technologies than Simple Object Access Protocol (SOAP) for communication between client and server because REST technology is more robust and leverages less bandwidth making it more suitable for internet usage. And also in future, we hope to develop a website including these mobile application functionalities. REST facilitates the use of server functionalities in the website more easily than SOAP.

### Business Layer

All the business logic is handled in this layer. For the ease of maintenance, this layer implements all the five main functionalities as completely separate five modules. Singing skill evaluator, chord generator, accompaniment music generator, pitch detector and audio workstation are the main components. Figure 4.3 illustrates the abstract view of the business layer.

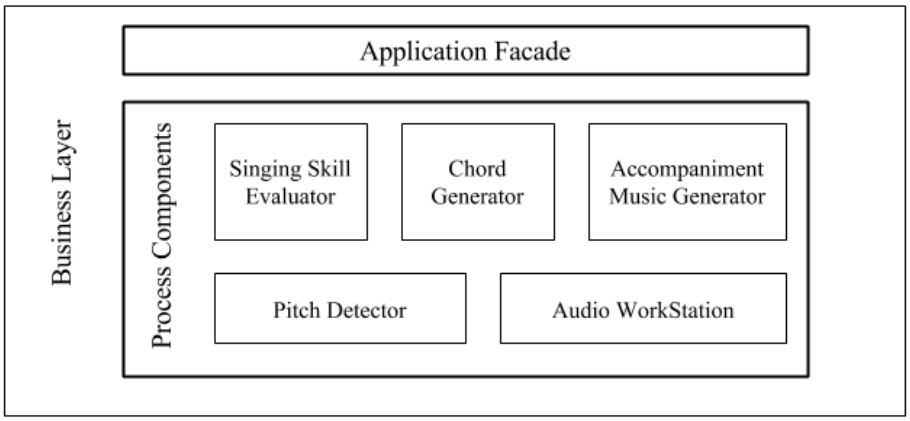



Figure 4.3: Abstract view of business layer

This application consists of five software components (subsystems). Facade design pattern enables a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystems easier to use. Therefore, facade design pattern is employed in building the business layer.

#### Data Layer and Cross Cutting Layer

Data layer communicates with the file system to store and retrieve mp3 files and also to access the trained Hidden Markov Model. Cross cutting layer is responsible for the API security which uses API keys for authorization.

## 4.3 Software Development Life Cycle

Software Engineering is defined as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" in the ISO/IEC/IEEE Systems and Software Engineering vocabulary [15].

Agile methodology was followed as the systematic approach to this project while following best practices and standards, turning to a more disciplined way. Version controlling and quality assurance methods were carried throughout the project to ensure it was inclined to a quantifiable approach. GitLab[1] was used for repository management and version controlling. Trello[2] was used to facilitate project management and issue tracking inclined to SCRUM which is an 'Agile Framework'. These methodologies were used in development, operation, and maintenance of the system which follows the definition of software engineering.

## 4.4 Summary

This chapter presented the overall design of the project. It began by laying out the main phases: vocal melody recording, pitch detection, singing skill evaluation, chord generation, accompaniment music generation and the combination of the recorded vocal melody and generated accompaniment track. Next, the chapter looked at the architectural design of the system which used a hybrid mobile application architecture where business and data layers reside in both the mobile device and the server. The chapter concluded describing the software development life cycle under agile methodology.

[1]https://about.gitlab.com/

[2]https://trello.com/

# Chapter 5

# Singing Skill Evaluation

Chapter 4 presented the overall design of the study. This Chapter describes the design of the singing skill evaluation under two categories. Section 5.1 lays out the design of the pitch based singing skill evaluation and Section 5.2 describes the design of the tempo based singing skill evaluation.

## 5.1 Pitch Based Singing Skill Evaluation

As mentioned in Chapter 2, most of the singing skill evaluation based research was carried out assuming the score information is available but this research focuses on the evaluation of the singing skill of unknown melodies. Therefore a classification method based approach was used. Pitch interval accuracy based features were extracted since pitch interval is known as a feature independent of a particular singer or a melody. The basic idea behind this approach is to measure the deviation within a semitone. Singing can be classified as good if the deviation seems to be stable and poor otherwise. Figure 5.1 illustrates the overview of the suggested approach. As the first step, the fundamental frequency $F0$ is estimated. Then smoothing and silent section removal is performed. Pitch interval accuracy estimation is carried out next and the term 'Semitone Stability' is defined using pitch interval accuracy. Once the semitone stability is computed, features are extracted. Finally, binary classification using two classes is performed.

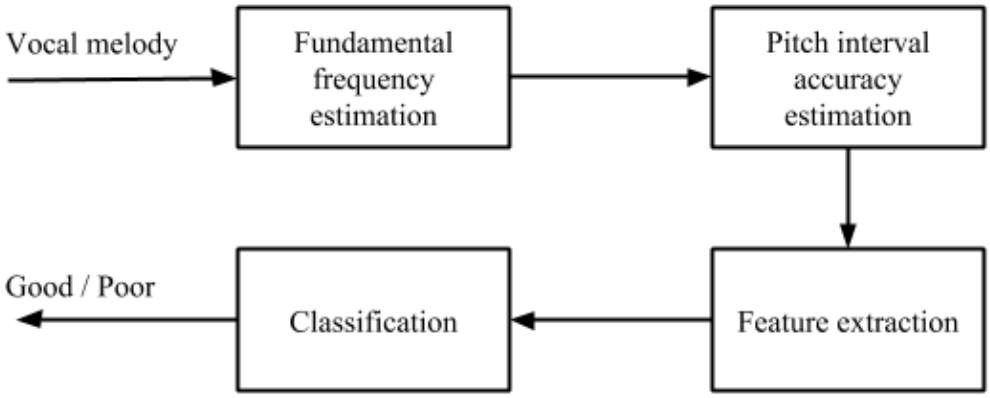



Figure 5.1: High level design of pitch based singing skill evaluator

### 5.1.1 Fundamental Frequency ($F0$) Estimation

The fundamental frequency is the lowest frequency component or partial which relates most to other partials[16]. Even though fundamental frequency and pitch are not the same, they are highly related since pitch values are calculated from log of $F0$ values. Therefore $F0$ estimation is essential in pitch based singing evaluation.

Low pass filtering is applied with an FIR low pass filter with 20 Hz cutoff frequency[8]. low pass filters are capable of creating a smoothing effect and filters out high-frequency components of the recording while removing the fluctuations such as vibrato and overshoot. Silent sections in the $F0$ trajectory were then removed.

### 5.1.2 Pitch Interval Accuracy Estimation

Pitch interval accuracy is known as an acoustic feature which is independent of a particular singer or a melody. It makes pitch interval accuracy an ideal feature for singing skill evaluation of unknown melodies.

According to the study of Nakano and Goto [8] pitch interval accuracy is measured by the fitting of the fundamental frequency trajectory to a 100 cent (semitone [1]) width grid. A particular pitch x can be mapped to a semitone width grid border(x is a multiple of 100) or between two borders(x is not a multiple of 100). In the first case where x is a multiple of 100, offset F is 0 and when x is not a multiple of 100 offset F is $(0 < F < 100)$ from its nearest lower border. Conversion of Hz values to cents was done using equation 5.1.

$$f_{cent} = 1200 \log_2 \frac{f_{Hz}}{440 \times 2^{\frac{3}{2}-5}} \tag{5.1}$$

[1]semitone is chosen based on the western music twelve-tone equal temperament. It divides an octave into twelve similar parts on a logarithmic scale which is known as a semitone. a semitone consists of 100 cents.

Next step is to compute semitone stability. Semitone stability is defined based on pitch interval accuracy and measures the deviation within a semitone.

Semitone stability $P_g(F,t)$ of time $t$ and width $T_g$ is computed as given in equation 5.2.

$$P_g(F,t) = \int_{t-T_g}^{t} p(F_{F0}(\tau);F)P_{F0}(\tau)d\tau \tag{5.2}$$

$(P_{F0}(t))$is the $(F0)$ possibility at time t and$(p(F_{F0}(t);F))$ is the gaussian comb filter for fundamental frequency $F0$ and offset $F$ at time $t$. gaussian comb filter for pitch $x$ and offset $F$ is defined in equation 5.3.

$$p(x;F) = \sum_{i=0}^{\infty} \frac{\omega_i}{\sqrt{2\pi}\sigma_i} exp\left\{-\frac{(x-F-100i)^2}{2\sigma_i^2}\right\} \tag{5.3}$$

Given below in Figure 5.2 are the graphs of $p(x,F)$ where $x = 4000,.....6000$ and $F = 1, 10, 75$ respectively.

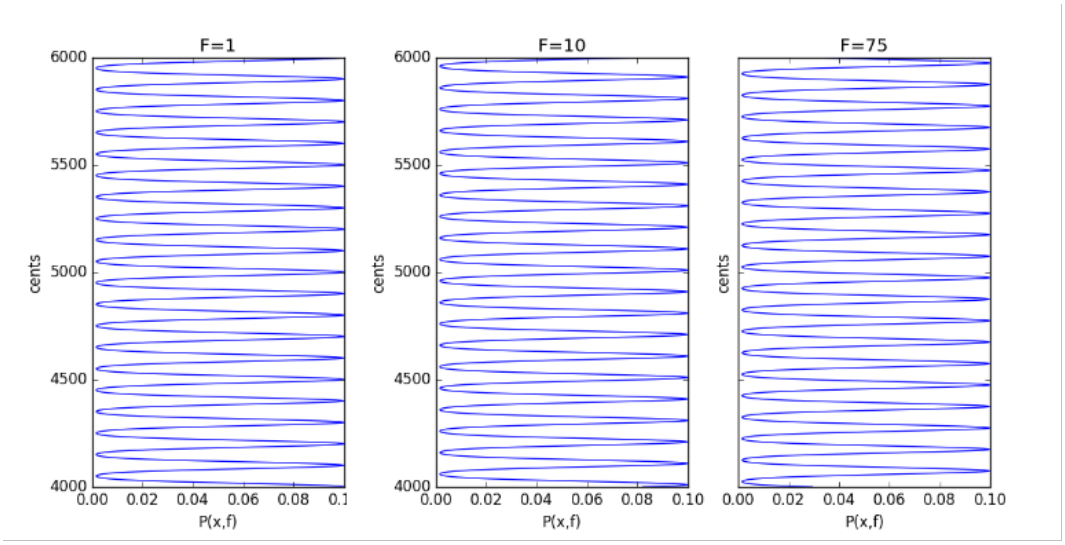



Figure 5.2: Gaussian comb filter for $F = 1, 10$ and 75

The process of semitone stability computation is described here. $F0$ trajectory was sampled into frames and semitone stability was computed for each frame. A frame must consists of 200 points of 10 millisecond samples. Each frame is shifted by 10 samples to get a new frame. Frame generation process is shown in Figure 5.3. For each frame, gaussian comb filter for offset $F = 0, 1, 2, .....99$ was applied and semitone stability of each frame was calculated.

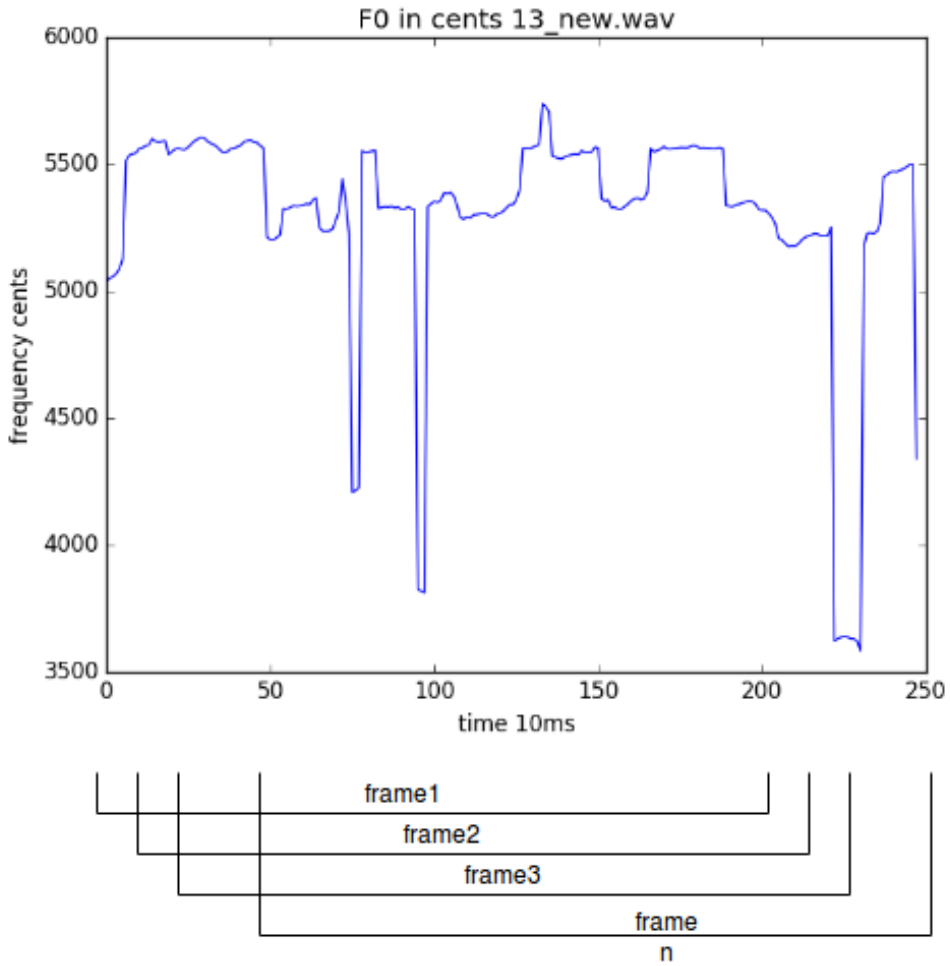



Figure 5.3: Frame generation of a song

Some frames obtained from semitone stability computation are shown below in Figure 5.4.

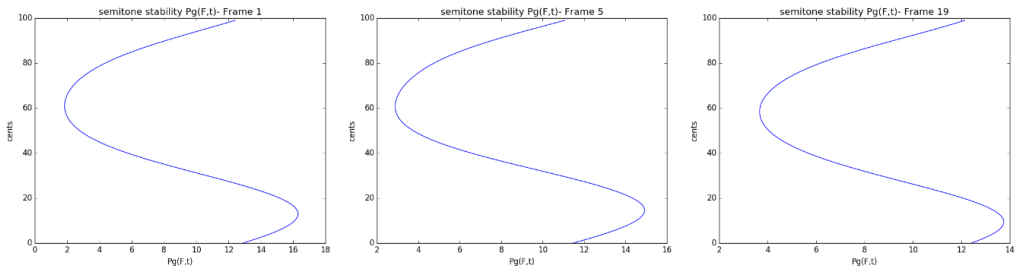



Figure 5.4: Semitone stability of frames

$Fg$ (grid frequency) is the frequency value which gives the maximum $Pg(F,t)$as given in Figure 5.4. In the Figure 5.5 given below,$Fg = 8$. (which means this frame fits best to a semitone grid where offset is equal to the grid Frequency.)

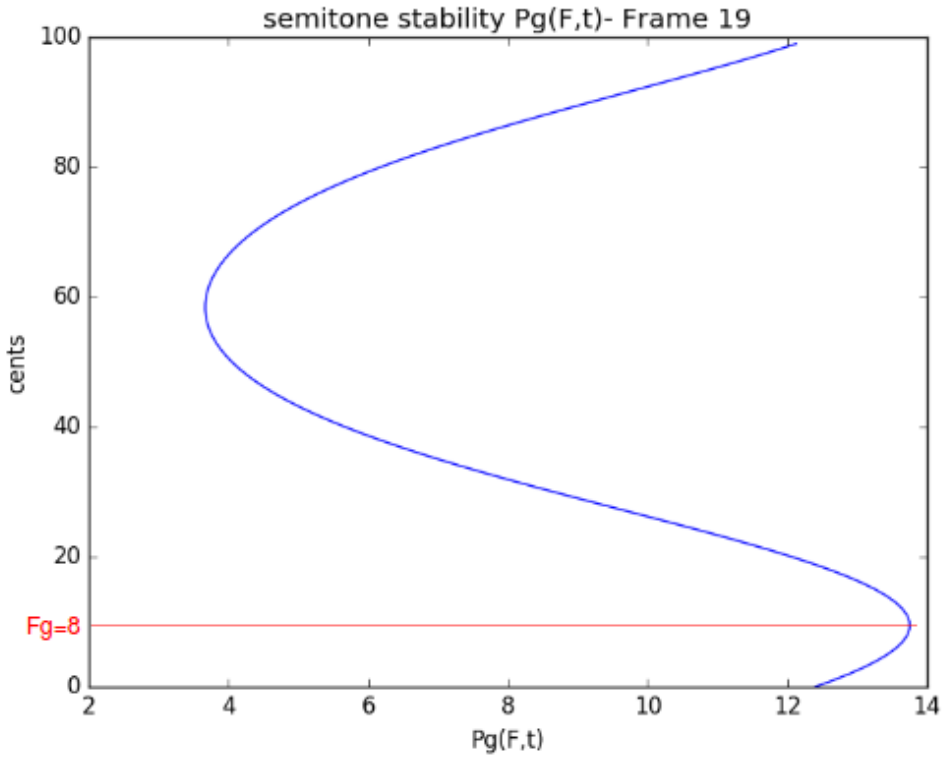



Figure 5.5: Semitone stability and the grid frequency of a frame

$$F_g = \underset{F}{argmax}\ P_g(F,t) \tag{5.4}$$

Once the semitone stability of all the graphs were computed, long term average $g(F)$ of those frames were taken. Since good singing should have a stable deviation within a semitone throughout the song a single sharp peak can be expected. Long term average of both good and poor singing are shown below in Figure 5.6.

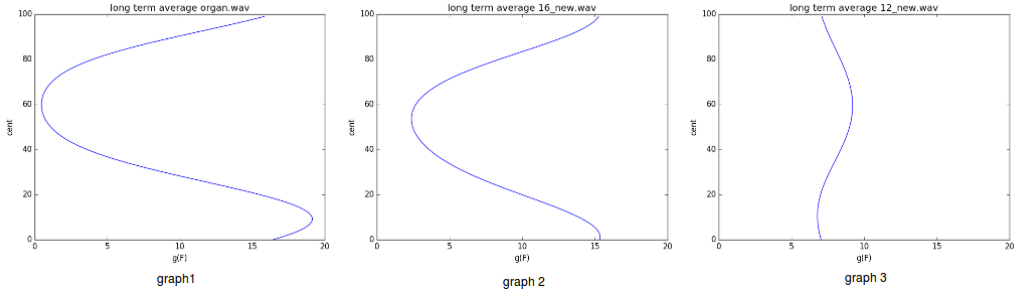



Figure 5.6: Long term average of selected melodies

Figure 5.6 first graph shows the long term average of a song played by an organ which has the sharpest peak among all three graphs. Figure 5.6 second graph shows the long term average of a good singing which has a single sharp peak but slightly less than the first graph.Finally Figure 5.6 third graph shows the long term average of a poor singing which has the least sharp peak compared against the other 2 graphs. Long term average graphs

can be explained further using grid frequency as given below. Grid frequency can be defined for the long term average $g(F)$ as given in equation 5.5.

$$F_g = \underset{F}{argmax}\ g(F) \tag{5.5}$$

If singing is good, there should be a stable deviation throughout the song. This deviation(offset) is given by the $Fg$ (grid frequency). Therefore an ideal graph can be presented where each and every frame has the same $Fg$ hence a single sharp peak at the grid frequency $Fg$. This ideal graph can be used to compare the actual singing with the ideal one. Figure 5.7 shows ideal and actual long term average of three different melodies.

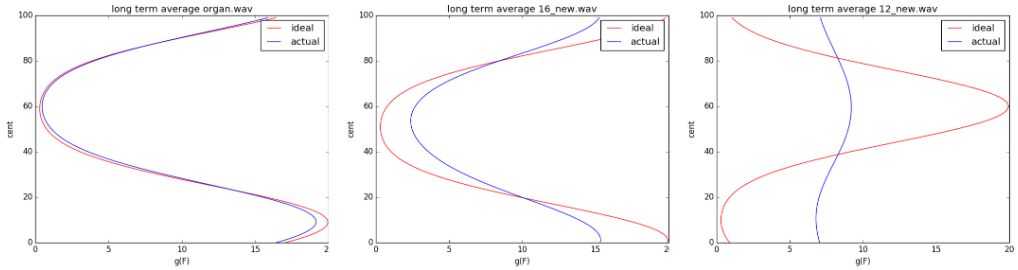



Figure 5.7: Ideal and actual long term average of selected melodies

### 5.1.3 Feature Extraction

As discussed in latter Section 5.1.2, followed by the computation of long term average $g(F)$, two features related to sharpness namely M value and slope value were obtained. M value(second moment) is the variance of the long term average of long term average curve as defined in equation 5.6.

$$M = \int_{F_g-50}^{F_g+50} (F_g - F)^2 g(F) dF \tag{5.6}$$

In order to compute the second feature(slope value), G(F) is defined as given in equation 5.7.

$$G(F) = \frac{g(F_g + F) + g(F_g - F)}{2} \tag{5.7}$$

Then slope of the linear regression line of the $G(F)$ function was taken

### 5.1.4 Classification

Above suggested features were extracted from sample melodies and they were labeled as either good or poor with the aid of a domain expert and given as an input to a SVM for a binary classification classified as good or poor. SVM s perform better when binary classification is to be performed and the amount of data is not high (around 100 data samples were used)[17].

## 5.2 Tempo Based Singing Skill Evaluation

The sense of timing is essential for musical related activities. Tempo is the speed of music that is measured using a number of beats per minute. Better timing provides better synchronization between accompaniment music and vocal melody. Because of that, tempo evaluation process is conducted to show, how a recorded vocal melody aligns with accompaniment music. As mentioned in Chapter 2, tempo evaluation of human voice is a hard task for typical tempo detection algorithms due to the variations and instability of voice. Vibrato suppression based super flux approach is chosen as the basis for onset detection due to this reason and Figure 5.8 brings out instability and variation of human voice over a guitar recording. Peaks were picked from the extracted onsets and tempo was calculated using the median value of peaks.

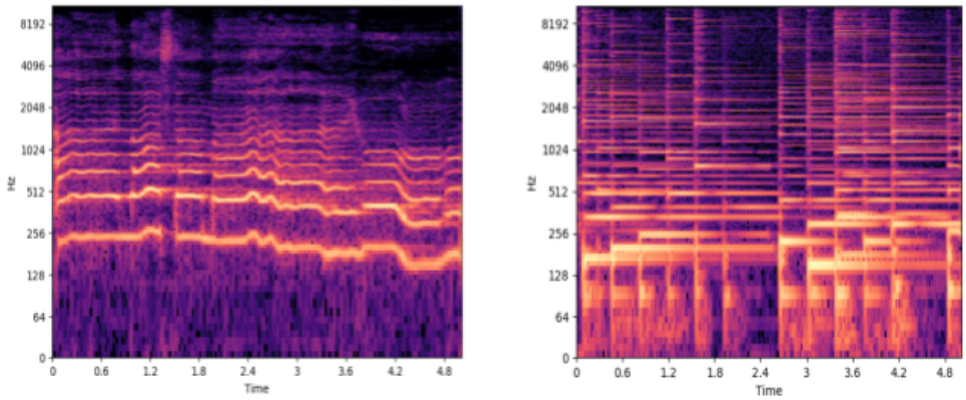



Figure 5.8: Spectrogram view of human singing and guitar recordings

Tempo detection algorithm involves three main stages: a front-end analysis for an efficient extraction of onsets from an audio signal, identify periodicity detection blocks and peaks to extract tempo, tempo tracking and disagreement fixing of manually annotated tempo with identified tempo. Figure 5.9 shows an overall architecture of tempo estimation process.

### 5.2.1 Onset Detection

Onset indicates the salient features of an audio signal. Onset detection method depends on the features of the audio signal. As mentioned in Chapter 2, Bock et al. presents vibrato suppression based onset detection method that enhances spectral energy flux based onset detection[12]. According to that study, energy-based methods are not the best choice for the onset detection of a vocal melody that contains soft onsets which has slow rise energy with long attacking time.

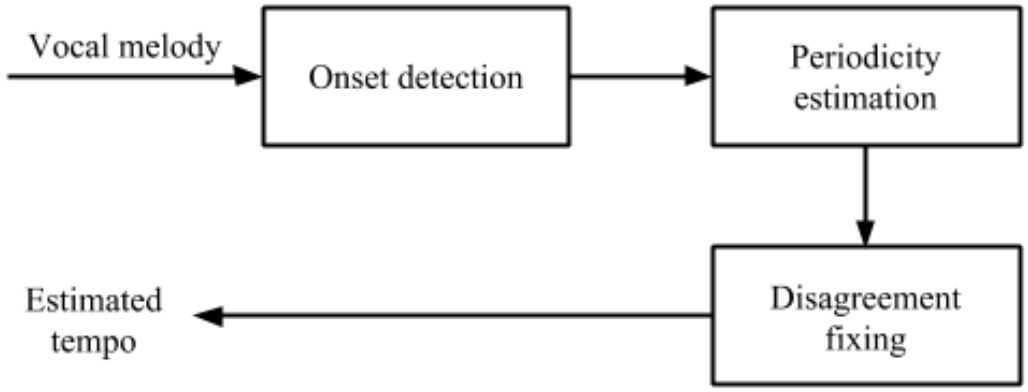



Figure 5.9: High level view of tempo estimation process

As the first step of vibrato suppression based onset detection algorithm, an audio signal is divided into overlapping chunks with Hann window function. Then the divided audio signal is converted into a decimated version of the Discrete Fourier Transformation (DFT). According to the range of window, the DFT is computed for points $X0, ...Xn-1$ ($n$ size window). Then again the DFT is computed for points $X1, ...Xn$. The process continues until it covers the whole signal[18].

According to that aforementioned study, they showed a Short Time Fourier Transformation(STFT) function to compute this STFT matrix directly. Because of that, STFT is used as a conversion mechanism of an audio signal with Hann window function in this project.

There is a special trajectory tracking stage in this algorithm. Some preprocessing steps are needed to facilitate this tracking stage. In preprocessing phase, a standard frame rate is doubled to increase the reporting accuracy and it quantizes magnitude features into much smoother trajectories. A vocal signal shows short time power spectrum and Mel Frequency Cepstral Coefficient based technique can accurately represent this envelope[19]. Because of that MFCC based mel scaled measures are used to produce smoother trajectories at this stage.

Common spectral flux calculation is modified by finding the difference between frames that are further apart (according to defined offset) instead of finding the difference between consecutive frames. Maximum filters were applied to widen the trajectory. As the final step, the difference of bins were calculated with respect to the maximum filtered spectrogram.

A peak picking phase has been employed in the separation of onsets and non onsets from maximum filtered spectrogram. According to that vibrato suppression onset detection study, if any point fulfills the following three conditions (5.8,5.9,5.10) then that point is selected as an onset of this signal.

$$SF*(n) = max(SF*(n - preMax : n + postMax)) \tag{5.8}$$

$$SF*(n) \geqslant mean(SF*(n - preAvg : n + postAvg)) + \delta \tag{5.9}$$

$$n - n_{previousOnset} > combinationWidth \tag{5.10}$$

$\delta$ - is a tunable threshold. preMax, postMax, preAvg, postAvg, comibinationWidth are depend on the frame rate of the audio signal. Figure 5.10 shows example of a picked set of onset of vocal melody after processing super flux algorithm.

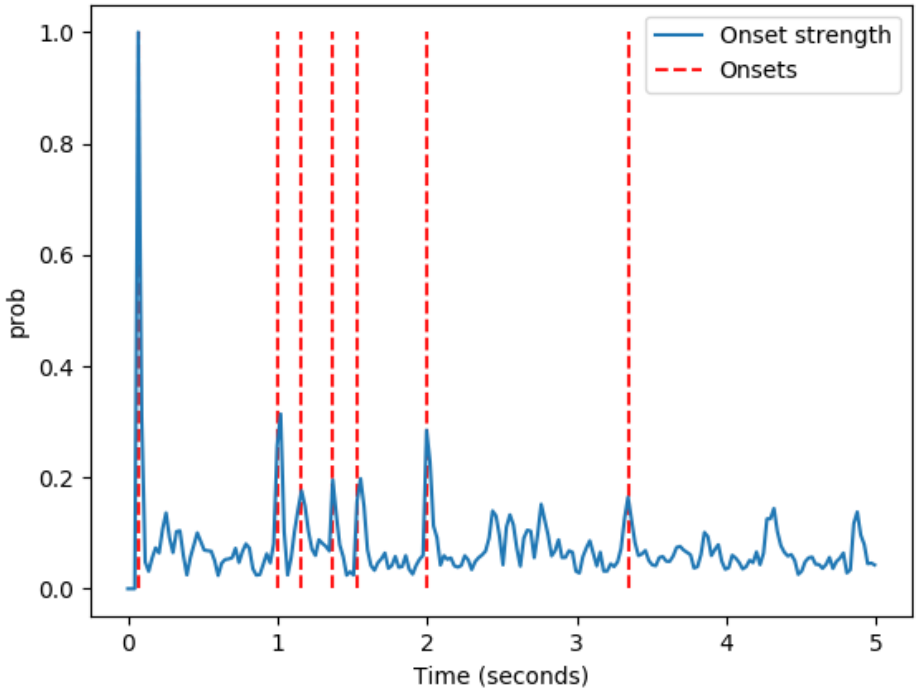



Figure 5.10: Onset strengths and selected onsets of an audio signal

### 5.2.2 Periodicity Estimation

As mentioned in Chapter 2, Alonso et al. proposed an approach for onset detection based tempo extraction using the concept of spectral energy flux[9]. But energy based onset detection methods are not well suited for vocal melody because of human voice like soft onsets shows instability and soft variation of the audio signal. Therefore onset detection process of this application is conducted through previously explained vibrato based super flux algorithm. But periodicity estimation of this spectral flux algorithm can be used to estimate periodicity of extracted onsets.

According to that mentioned study, there are two traditional methods that estimate periodicity of onsets: Spectral product and autocorrelation function. Spectral product based

technique assumed that the spectrum of the signal is formed from strong harmonic locations. Because of that, Spectral product based technique is not well suited for periodicity estimation of extracted soft onsets.

Autocorrelation function provides a correlation of the signal with a delayed copy of that signal itself. It is good for periodicity estimation of soft onset because it reveals regular periodic structure based on patterns. Tempo is estimated by analyzing lag of the larger peaks in autocorrelation and using the multiplicity of the relationship among them. If there is no relation among peaks, the tempo is estimated using only a lag of the largest peak. Figure 5.11 shows an example curve which is created by mapping autocorrelation points of an audio signal.

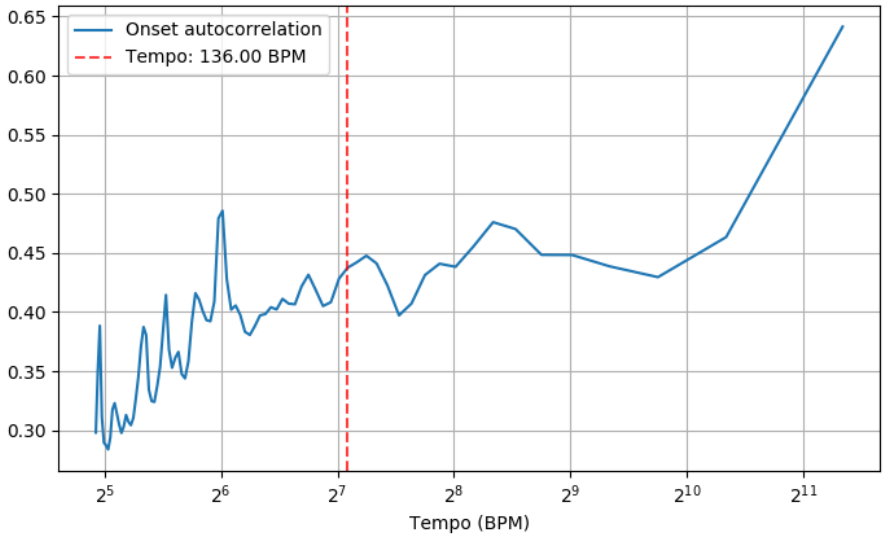



Figure 5.11: Onset autocorrelation curve and tempo estimation of the audio signal

### 5.2.3 Disagreement Fixing

As the final step of tempo detection process, the disagreement of manually annotated tempo with identified tempo should be fixed. There is a very common problem when human taps along with music to identify tempo and automatic tempo estimation methods also make this doubling or halving of the true tempo. Alonso et al. introduced the 5.11 mathematical equation to address this disagreement issue[9].

$$0.95\alpha T < T_r < 1.05\alpha T \;\; with \;\; \alpha \in \{0.5, 1, 2\} \tag{5.11}$$

Tempo estimation T provided by the algorithm is labeled as correct if there is a less than 5% disagreement from the manually annotated tempo used as reference $Tr$ according the above expression.

## 5.3 Summary

This chapter described the design of the singing skill evaluation under two categories: pitch based singing skill evaluation and tempo based singing skill evaluation. Deviation within a semitone is measured in pitch based singing skill evaluation. First, the fundamental frequency is extracted from the vocal melody and pitch interval accuracy, judged by the fitting of fundamental frequency trajectory to a semitone width grid, is estimated, features related to pitch interval accuracy are extracted and classified as good/poor by an SVM classifier. The alignment of recorded voice melody with the generated accompaniment music is evaluated in tempo based singing skill evaluation. As the first step, an onset of the vocal melody is detected using Vibrato suppression based super flux approach. Then periodicity of onsets is estimated with autocorrelation function and disagreement of manually annotated tempo with identified tempo is fixed as the final step of the tempo evaluation process.

# Chapter 6

# Harmonization of Vocal Melodies using HMM

In Chapter 5 we presented the overall design of this study and in Chapter 6 we elaborate more on singing skill evaluation method. This Chapter describes the process of harmonization of vocal melodies using a Hidden Markov Model in detail. Section 6.1 outlines the integration of HMM to music to generate chord sequences and the design assumptions. Section 6.2 presents the training and the building phase of the HMM and Section 6.3 describes the method of generating chords for a new vocal melody. The design of the harmonization of vocal melodies is solely based on the methodology which is used by MySong (Currently known as SongSmith)[1, 2].

## 6.1 Design Overview

It is important to note that a single melody can have different chord sequences. The variance of these chord selections is dependent on the musician and the genre of the melody. Therefore the goal of automatic harmonization of vocal melodies is not to obtain the correct chord sequence but to extract more appropriate chord sequence from the melody. Figure 6.1 represents the overall design of the chord generation phase.

A vocal melody can be represented as a sequence of musical notes and they can be in different octaves. However, in this model, an assumption has been made to simplify the process of harmonization stating that the octave informations was not relevant for selecting chords for a melody. A vocal melody in this study is thus a sequence of notes in which each element corresponds to a specific "pitch class": a set of all pitches corresponding to one of the standard 12 tones in the chromatic musical scale which is independent of octave displacement (e.g. C, C#, D, Eb, etc.). In the training phase, this sequence is obtained from published musical scores (see Section 6.2) whereas in the decoding phase this sequence is obtained by sampling the user's recorded vocal melody (see Section 6.3).

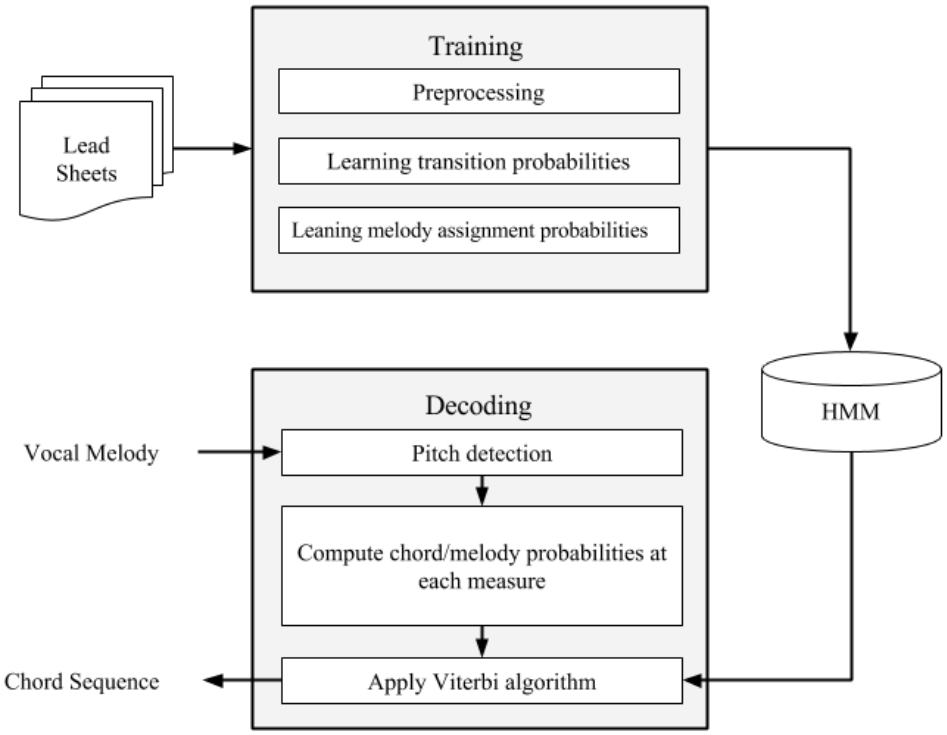



Figure 6.1: Overall design of the chord generation phase

### 6.1.1 Hidden Markov Model

This Section explains the adaptation of Hidden Markov Model to music concepts. HMMs are typically used to model statistical machine translations. Chord generation for a vocal melody can be thought of a model of natural language translation. Language translation models translate sequence of words of one language to sequence of words of another language. These models are also known as sequence to sequence models and HMMs are one of the basic techniques to model them. Sequence to sequence models can specially model the dependencies between states. The words of natural language sentences are dependent of the previous words. We can see the same characteristic in chord progressions of melodies. The equivalent form for words in natural languages in our model is measures which contain a sequence of pitch classes. Translation of these measures to chords is the key idea to use an HMM. We assume the reader is familiar with Hidden Markov Models. Readers unfamiliar with HMMs are referred to Rabniers's tutorial[20] for an overview.

Figure 6.2 depicts the Hidden Markov Model representation of our model. Each measure corresponds to a single node and the state (the chord selected to be played during the measure) is initially unknown. The observations are the vectors of probabilities which are constructed using the sampled melody which is described in Section 6.3.1 Transition probabilities among states are estimated from training data (see Section 6.2.2). The hidden states are the chords and the best sequence of chords are derived from applying Viterbi algorithm described in Section 6.3.3.

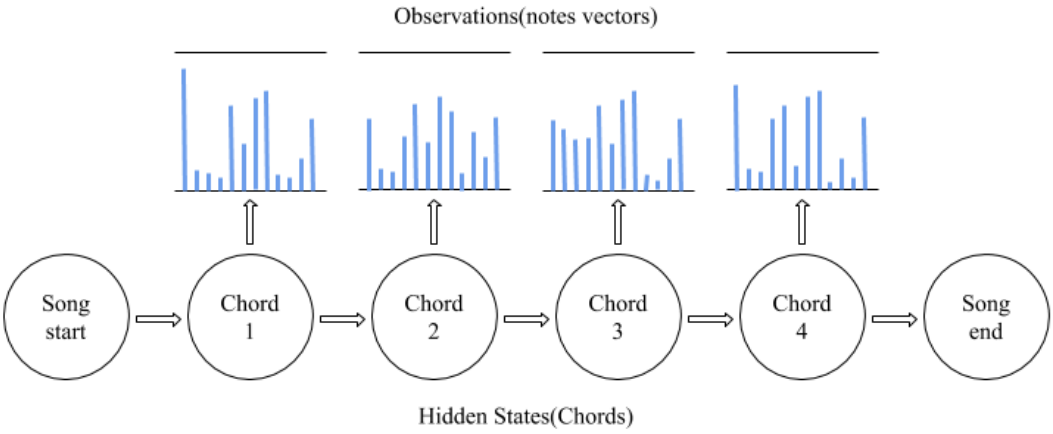



Figure 6.2: Hidden Markov Model representation of the chord generation process

### 6.1.2 Assumptions

The following assumptions have been made so that the output of the chord generation phase is acceptable only if these assumptions are held.

1. One measure from the vocal melody can have only one chord per that measure. Here we use the term "measure" to refer the smallest amount of time for which a chord will be played.
2. A vocal melody can only have 12 possible pitch classes. All notes can be reduced to a single octave without losing the information that is needed for chord generation.
3. A chord for a measure is only dependent on the fraction of the measure during which each pitch class is heard and the chord of the previous measure.
4. The musical key of the melody does not change within melody.
5. The rhythm pattern and tempo of the melody does not change within the melody.

## 6.2 Training

The training of the Hidden Markov Model has been done using a database of 289 lead sheets, each of which contains a melody and the corresponding chord sequence. The lead sheets contain popular western songs ranging from pop, rock, R&B, jazz and country music. In this study we did not implement the training phase and re-used the trained model from SongSmith, hence we briefly describe the method of training only.

### 6.2.1 Preprocessing

The variety of chord kinds is extremely large and treating all of these chords independently require a huge database of lead sheets. Major, minor, suspended, diminished and augmented

are the five basic triad chord kinds in music which contain three notes in each chord. All the other chord kinds in the database were simplified to these five core triads and assumed that appropriateness of the converted chords did not change for a particular measure.

Songs can be classified into one of 12 musical "keys": a key represents a distribution of frequently occurring notes and chords. Key information was available in each of the lead sheets and all of the lead sheets were transposed to a single key (key of C).

### 6.2.2 Learning Transition Probabilities

The hidden states of the HMM are the chords in this model. Transition probabilities defines the probability to which a states transit from another state or in our model it is the probability to which a chord transit from another chord and can be mathematically defined as $P(c_i|c_{i-1})$. Since there are 12 musical keys and 5 core chords kinds there can be 60 possible chords or hidden states. Additionally, SongSmith has considered two more virtual chords as starting chord and ending chord. Including these, there are a total of 62 chord kinds in the database and a $62 \times 62$ matrix was constructed in which each cell represents the number of times a transition occurred between the corresponding two chords in the database. This matrix was referred to as the chord transition matrix. Each row of this matrix must be normalized so that we can have the probability of each possible transition.

Generally, the 12 musical keys occurs in one of two distinct modes: major or minor. The chord transitions and the emotional factor of the music change when the above mentioned mode changes. Because of that, the lead sheet database was divided into two clusters as major and minor and trained to obtain two transition matrices as major transition matrix and minor transition matrix. Section 6.3 uses these matrices to come up with a chord progression given the recorded vocal melody.

### 6.2.3 Learning Melody Assignment Probabilities

Melody assignment probabilities defines the statistics about which notes are associated with each chord type. Each musical note has a specific time fraction and the melody assignment probabilities were calculated by counting the total duration of each musical note occurring in the measure against each individual chord. Using all of these probabilities a $60 \times 12$ matrix was constructed where each row index of this matrix represents a chord kind and each column index of this matrix represents a pitch class. All the rows of the matrix was normalized and each element of the matrix represents the probability of seeing a pitch class given a chord kind. This matrix is referred to as the melody observation matrix and unlike transitions matrices, this matrix does not have a corresponding major or minor mode since the note fraction of a measure does not depend on the mode of the melody.

## 6.3 Decoding

Decoding phase is where the generation of chords for a new melody takes place. The vocal melody being analyzed was recorded with a consistent tempo and the timing information from the vocal track was not considered. The chords are generated at fixed intervals and this fixed duration is referred to as a measure. Decoding phase consists of several steps such as pitch detection, computing chord/melody probabilities and choosing the best chord sequence using the Viterbi algorithm. Following sections describe the steps of the decoding phase in detail.

### 6.3.1 Pitch Detection

The first task in harmonizing a vocal melody is to compute the pitch of the recorded vocal melody. These pitch values are calculated as fundamental frequency values using the method described in Section 4.1.2. Once these values ($f_{max}$) for the vocal melody have been extracted the continuous pitch class $p$ corresponding to $f_{max}$ is calculated as $p = 12\log_2(f_{max}/523.2)$. The offset($p_{offset}$) between $p$ and the nearest known musical pitch class is computed for all the sample points in each measure and the mean $p_{offset}$ is calculated. Then these mean $p_{offset}$ values are added to the value $p$ of each sample point so that the pitch sequence to approximately align with the standard chromatic scale. Finally, each $p$ value is rounded to the nearest integer $p_{int}$ and the final pitch class values are computed as $p_{int} \mod 12$. The histogram of these integer pitch classes at each measure is referred to as the observation vector for that particular measure.

### 6.3.2 Computing Chord/Melody Probabilities at Each Measure

The observation vectors from pitch detection phase and the rows of the melody assignment matrix are similar in nature. The likelihood of a chord for each measure is computed by taking the dot product of the observation vector with the appropriate row of the melody observation matrix. A list containing all these probabilities are stored and referred to as the chord emission probability matrix.

### 6.3.3 Choosing the Best Chord Sequence

This Section depends on the assumption that the melody that is being examined is in the key of C. However a melody can be in any key and later it will be shown how to generalize chord selection procedure for a melody in any key. Given the observations for the Hidden Markov Model, the most likely sequence of hidden states (chord kinds) can be generated using the Viterbi Algorithm [21].

Viterbi algorithm generates a path $X = (x_1, x_2, ..., x_T)$ which is a sequence of states $x_n \in S = \{s_1, s_2, ..., s_K\}$. This set $S$ corresponds to the set of all chord kinds in our model.

Other inputs to the Viterbi algorithm are $\Pi, Y, A$ and $B$ and they are corresponding to initial transition probabilities array, sequence of observation vectors,chord transition matrix and chord emission matrix respectively. Viterbi algorithm is dynamic programming algorithm and two 2-dimensional matrix of size $K \times T$ are constructed.The probability of the most likely path is stored at each element of $T_1$ and the most likely path is stored at each element of $T_2$. The procedure of the Viterbi algorithm is described in pseudo code in the following Algorithm 1.

**Algorithm 1** Viterbi Algorithm

**procedure** VITERBI($S, \Pi, Y, A, B$):$X$
  **for** each state $i \in \{1, 2, .., K\}$ **do**
    $T_1[i, 1] \leftarrow \pi_i \cdot B_{iy_1}$
    $T_2[i, 1] \leftarrow 0$
  **end for**
  **for** each observation $i \in \{2, 3, .., T\}$ **do**
    **for** each state $j \in \{1, 2, .., K\}$ **do**
      $T_1[j, i] \leftarrow \max_k(T_1[k, i-1] \cdot A_{kj} \cdot B_{jy_i})$
      $T_2[j, i] \leftarrow \operatorname{argmax}_k(T_1[k, i-1] \cdot A_{kj} \cdot B_{jy_i})$
    **end for**
  **end for**
  $z_T \leftarrow \operatorname{argmax}_k(T_1[k, T])$
  $x_T \leftarrow s_{z_T}$
  **for** $i \leftarrow T, T-1, .., 2$ **do**
    $z_{i-1} \leftarrow T_2[z_i, i]$
    $x_{i-1} \leftarrow s_{z_{i-1}}$
  **end for**
  Return $X$
**end procedure**

### 6.3.4 Key Determination

As described in the preprocessing phase all the songs in the database were transposed to a single musical key before training the Hidden Markov Model. Based on the assumption that the transitions matrices in each key are identical other than a simple circular shift of the transition matrix, each integer pitch class in the vocal melody is transposed up by one semi-tone each and the Viterbi algorithm is applied. Since there are 12 semi-tone steps in the standard chromatic musical scale, the Viterbi algorithm has to be run 12 times. At $k$ steps the overall likelihood of the optimal chord sequence generated from the Viterbi algorithm becomes the highest and the value $k$ is chosen as the candidate. The vocal melody being

examined can be in any of the 12 keys. While the melody is being transposed up, it is in the key of C when the number of steps is equal to $k$ and at that time the output from the Viterbi algorithm is taken. Finally, the generated chord sequence is transposed down by $k$ steps to get the original chords for the recorded melody.

### 6.3.5 Happy Factor

As described in the Section 6.2.2 two transition matrices were constructed based on the mode of songs as major transition matrix($P_{maj}$) and minor transition matrix($P_{min}$). A parameter $0 \leq \beta \leq 1$ referred to as the "Happy Factor" was used to control the contribution from each transition matrix. The mixing of the two transition matrices is given by the equation 6.1

$$P(c_i|c_{i-1}) = P_{maj}(c_i|c_{i-1})^{\beta} \times P_{maj}(c_i|c_{i-1})^{\beta-1} \tag{6.1}$$

The"Happy Factor" parameter was directly exposed to the user to gradually adjust the feel of an accompaniment from "happy" to "sad".

## 6.4 Summary

This chapter described the design of harmonization of vocal melodies and laid out the design overview of the chord generation phase which consists of a training and a testing phase. The key idea of chord generation for a vocal melody is to translate measures which contain a sequence of pitch classes into chords. A Hidden Markov Model was used as the core of the chord generation process. Each measure from the vocal melody is an observation for the HMM and corresponding chords are the hidden states of the HMM. Transition and Emission probability matrices were constructed from training data and the best sequence of chords was derived from applying the Viterbi algorithm.

# Chapter 7

# Implementation

In Chapters 4, 5 and 6 system design and design considerations were described. This Chapter contains the details of the implementation of components described in previous chapters. Section 7.1 describes tools and technologies that were used in the implementation phase of the project. Section 7.2 and Section 7.3 describe the implementation of the server and the mobile application respectively.

## 7.1 Tools and Technologies

This section provides a brief understanding to the tools and technologies used in this project with the relevant justifications for their selection.

### 7.1.1 Python

Python is a widely used high level and interpreted programming language which supports multiple programming paradigms including object-oriented, imperative, functional and procedural. Python is chosen as the implementation language of the business layer of the server side, because of the availability of comprehensive standard libraries and extensions of the advanced audio processing libraries.

### 7.1.2 Node.JS

Node.JS is a free and open source server-side framework which is built on JavaScript environment. Node.JS version 6 is the currently active long term support version. Non-blocking I/O calls is one of the main advantages of it. Node.JS is selected to implement the service layer of the server considering this advantage since it allows a large number of concurrent users to connect to the server and addresses the practicality aspects of the system by providing better performance through load balancing.

### 7.1.3 Android

Android is one of the main mobile operating systems in the world. Google has developed Android based on Linux kernel and is designed primarily for touchscreen mobile devices such as smart phones and tablets. The Java based native android development method was used to implement the mobile application because of the high availability of audio processing libraries.

### 7.1.4 Praat

Praat is a free software for scientific analysis of speech in phonetics. This software was built using C++ programming language and can be run on Unix, Linux, Mac and Microsoft Windows environments. Praat provides a better approximation for fundamental frequencies($F0$) of the given audio signal by filtering out the noises effectively. Therefore, Praat is used as the pitch detection component to extract fundamental frequencies($F0$) from a recorded vocal melody. A better approximation of fundamental frequency is needed for chord generation and pitch based singing skill evaluation processes which can be satisfied with Praat.

### 7.1.5 Python Librosa

LibROSA is a free and open source Python-based music and audio analysis library. It provides customizable digital signal processing functions and music information retrial functions for audio signal analyzing. Both of these digital signal processing libraries and music information retrieving functions are needed for a tempo estimation process of vocal melodies. LibROSA was used to implement tempo estimation because of direct information retrieval process and ease of making a consistent flow between functions of the same library.

### 7.1.6 Python NumPy

NumPy is a powerful Python library introduced for scientific computing. Most of the python packages like SciPy, libROSA use NumPy for internal mathematical calculation and scientific computing purposes. There are lots of data manipulations and calculation processes in the server-side business layer. NumPy provides powerful operations like array-based operations, linear algebra and mathematical calculation. Because of the efficiency and simplicity, server-side calculations and scientific computations were implemented using NumPy.

### 7.1.7 MMA

MMA stands for Music MIDI Accompaniment. It is a free music accompaniment generator which is written in python. It provides a MIDI file as an output according to a given chord sequences and MMA directives. MMA is used to generate a MIDI track in this project

because of compatibility, ease of use and free for use in any products under GNU [1] public license.

### 7.1.8 Timidity++

Timidity++ is a free software that can be run under Unix, Microsoft Windows and AmigaOS environments. This software supports many file types including MIDI, .kar(MIDI with karaoke) and module files. Timidity++ is used to read MIDI files and convert them into corresponding WAV files in this system because of the support and compatibility to apply different kinds of sound fonts which is responsible for generating the actual sounds written in the MIDI files.

### 7.1.9 Lame

LAME is a free software that converts an audio file to mp3. LAME is used to convert audio files within server before mixing and transferring processes because of high-quality encoding mechanisms and compatibility.

### 7.1.10 Python PyDub

PyDub is a free audio processing python package which facilitates segment wise audio processing. PyDub is used for audio editing and mixing processes in this project, because of compatibility concerns, effective mixing and editing processes without losing the quality of audio.

### 7.1.11 Sox

Sound eXchange (SoX) is a cross-platform(Linux, Windows, Solaris, OS X) and free software for audio editing. It is written in standard C language. In this project, Python wrapper for SoX is used for audio filtering and effect adding processes to increase the quality of the output song.

### 7.1.12 Python scikit-learn

scikit-learn provides many tools which facilitate classification, regression, clustering, model selection and preprocessing. In this project, scikit-learn is used for classification using an SVM, cross-validation under model selection and linear regression because of the simplicity and effectiveness of data mining and analysis in python environment.

[1] GNU general public license is a free software license, which guarantees users to run, share, modify software without restrictions.

## 7.2 Implementation of the Server

The server is composed with three layers separating the logic and each of these layers were implemented using different technologies. The implementation of the business layer is elaborated more since it is composed with core components of this system.

### 7.2.1 Service Layer

All the HTTP requests from the mobile application are handled by the service layer and this layer was implemented using Node.js (see Section 7.1.2). The express.js framework for Node.js has been used to implement the RESTful nature of the application. Each REST call is secured using JSON web tokens and Google authentication was used to authenticate the users of this system. The application facade functions of the business layer are called during a REST request from the mobile application and the output from the business layers is again sent back to the mobile application through the service layer as the response to the request.

### 7.2.2 Business Layer

Business Layer performs the core functionalities of this system. This layer was implemented using Python 2.7 (see Section 7.1.1) with object oriented programming model. The layer was composed of five components and those components were structured using python classes. The component based model was used considering the abilities of re usability and maintainability. Figure 7.1 represents the interactions within the components to generate an accompaniment music for a recorded vocal melody.

The vocal melody is first taken as an input to the pitch detector. The pitch detector component outputs estimated fundamental frequency values for the recorded vocal melody. These values are then used by the singing skill evaluator and the Chord Generator components for further processing. Singing Skill Evaluator component sends the feedback about the skill of the recorded vocal melody to the user while the Chord Generator component performs the selecting of appropriate chord sequence using the trained HMM and outputs the chord sequence. The chord sequence, rhythm style, and the predefined tempo are taken as inputs to the Accompaniment Music Generator and the corresponding accompaniment music for the recorded vocal melody is generated. Finally, the vocal melody and the generated accompaniment music is combined using Audio Workstation component to obtain the complete song.

A detailed description is presented in the following sections regarding the implementation of components in the business layer.

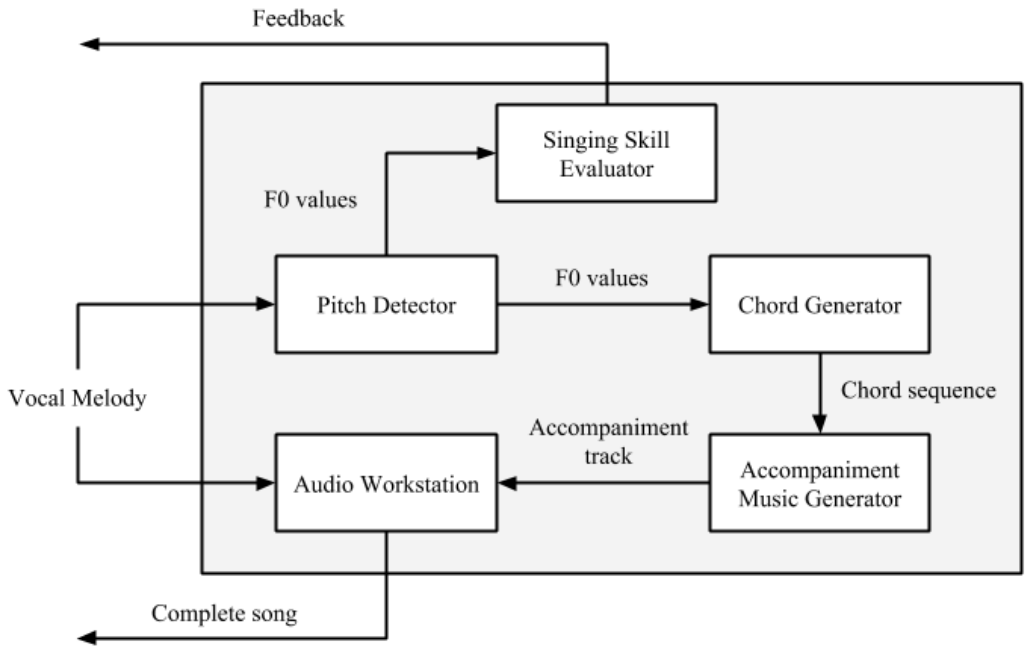



Figure 7.1: Interaction of components within accompaniment music generator

### Pitch Detector

Pitch detector component was implemented as a wrapper class for the Praat (see Section 7.1.4) "To Pitch" method. The Praat program was called as a subprocess through a Praat script since Praat does not provide a command line interface. The interaction between the Pitch detector python class and the Praat program is represented in Figure 7.2.

The praat script can be executed from the Pitch detector class to extract the fundamental frequency ($F0$) from an audio file with low pass filtering applied. Pass Hann Band filter was used as the low pass filter with 5 Hz cutoff frequency.

```
Filter (pass Hann band)... 0 1000 5
```

The following Praat script line shows the $F0$ extraction using 'To Pitch' command in praat and it will return pitch list with the time. To Pitch (ac) method performs a pitch analysis based on an autocorrelation method described in Section 4.1.2. The settings of the To Pitch method is set to its standard values. The Time step is set to 0.0 so that the Praat will use a time step of 0.75 / (pitch floor) hence calculates 100 pitch values per second. The Pitch floor is set to 75 Hz and the candidates below this frequency will not be recruited and the value 15 indicates the maximum number of candidates to be recruited. The Pitch ceiling is set to 600.0 Hz and the candidates above this frequency will be ignored. The complete praat script is shown in the Appendix(see appendix C.2)

```
ToPitch(ac)...0.0 75.0 15 off 0.03 0.45 0.01 0.35 0.14 600.0
```

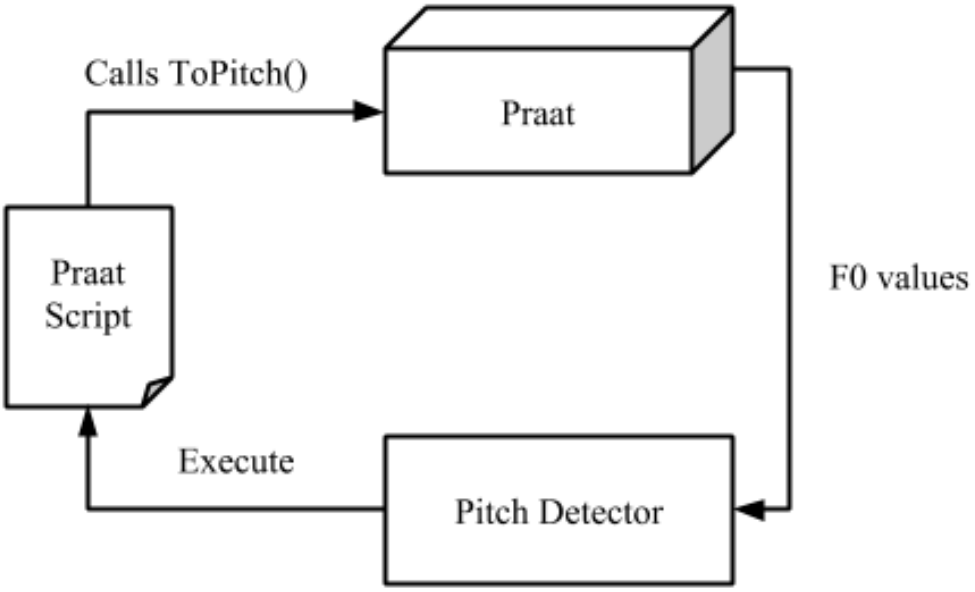



Figure 7.2: Interaction between pitch detector and Praat program

**Pitch Based Singing Skill Evaluator**

Python 2.7 was used for the implementation of pitch based evaluator. Implementation is described under fundamental frequency estimation, pitch interval accuracy estimation, feature extraction and classification as good/poor. Numpy, sklearn, scikit libraries were used for the implementation.

Fundamental Frequency $F0$ was extracted using pitch detector, described in Section 7.2.2. Pass Hann Band filter was used as the low pass filter with 20 Hz cut off frequency.

Silent sections were removed from the frequency values obtained from the pitch detector.

As described in Section 5.1.2, pitch interval accuracy estimation consists of two main steps, converting Hz values to cents and semitone stability estimation. Conversion of Hz values to cent values were done using the equation 5.1 given in Section 5.1.2. Python numpy arrays were used to store frequency values and math library was used to implement the mathematical functions. Next step was computing semitone stability for each frame. Therefore gaussian comb filters for offset($F = 0, 1, 2, .., 99$) was applied for each and every frame where a frame has pitch values between(3000,6000) In order to improve the performance comb filter values were computed as required and stored in a two dimensional array so they can be reused.

```
pxf = [[(-1) for y in range(100)] for x in range(3000, 6000)]
```

Once the semitone stability for each frame is computed, long term average of all frames can be computed. M value(variance) and the slope value were computed as shown in Section 5.1.4. For the integration, simps library was imported from scipy.integrate. Functions for

computing above features are presented in the appendix (C.3),(C.4). Visualization of the behavior of the target classes with features M value and slope value is given below.

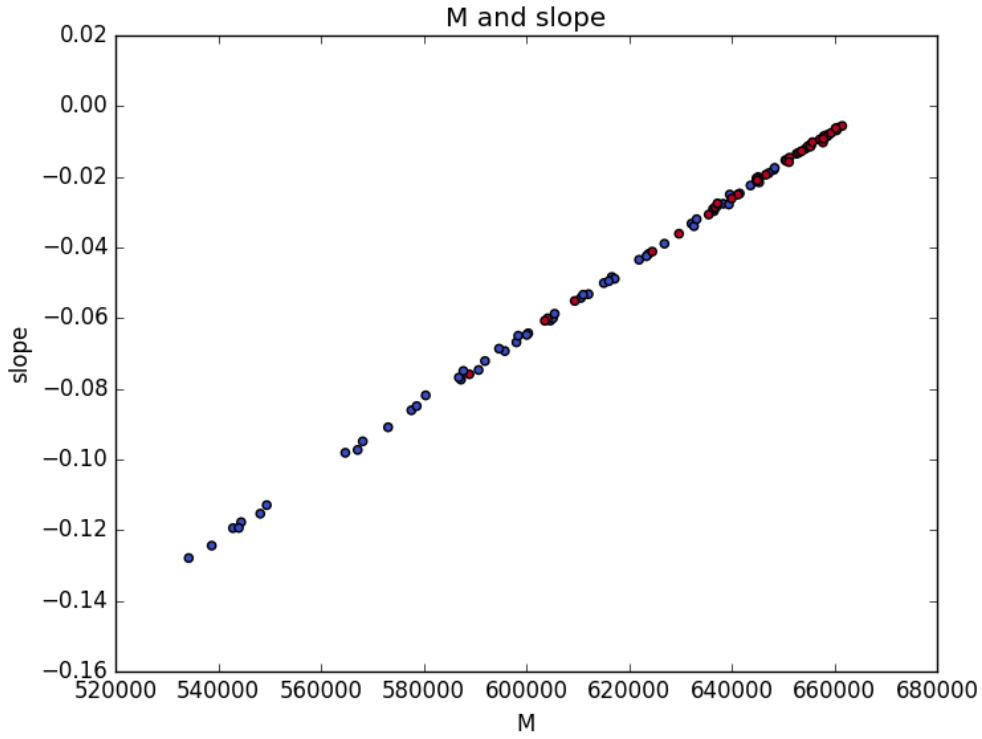



Figure 7.3: Behavior of target classes with M value and slope value

As shown in Figure 7.3 sample data we collected seemed to be linearly separable therefore an SVC (support vector classifier) SVM with a linear kernel was used as the classifier.

```
svc = svm.SVC( kernel='linear', C=1.0 )
```

Python Sklearn was used to implement the SVM as a solution to overfitting problem k fold cross validation was used. Sklearn model selection library provided KFold function.sklearn metrics library was imported to compute the metrics, accuracy, precision and recall.

### Tempo Based Singing Skill Evaluator

The proposed method for tempo-based evaluator was implemented using Python 2.7 (see Section 7.1.1) with the use of Python Librosa library (see Section 7.1.5) for audio and music signal analysis, Numpy (see Section 7.1.6) for linear algebra and Fourier transformations. In this section, the implementation details of the tempo based evaluation is described according to the sections discussed in Section 5.2: Onset detection, Periodicity estimation and Disagreement fixing.

As mentioned in Section 5.2.1, STFT is performed with the input audio signal to change the domain of signal. An audio signal is divided using overlapping chunk with Hann window

function while performing STFT.

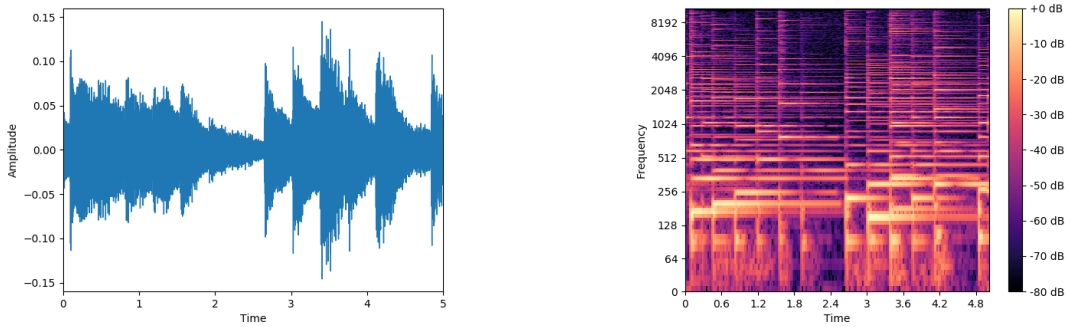
An audio file is loaded using librosa.load function. Floating point time series and sample rate of the audio file are returned as the output of this function.

```
time_series , sample_rate = librosa.load('<audio_file>')
```

STFT is done using librosa.core.stft function. Window function and window length can be passed as parameters for that function. Then it reruns STFT matrix of a given time series(signal).

```
stft_matrix = librosa.core.stft (time_series ,
                                 win_length=<n>, window='hann')
```

Figure 7.4 shows an example of time domain view of a signal before transform and short time Fourier transform view of the signal.




(a) Time domain view of signal

(b) STFT view of signal

Figure 7.4: Time domain and STFT view of signal

As mentioned in the design section, there is a trajectory tracking stage in this algorithm. The standard frame rate is doubled to increase reporting accuracy. Mel Frequency Cepstral Coefficients is used for this process. Few stages are needed to perform to get a mel scaled measures from an audio signal using MFCC based technique.

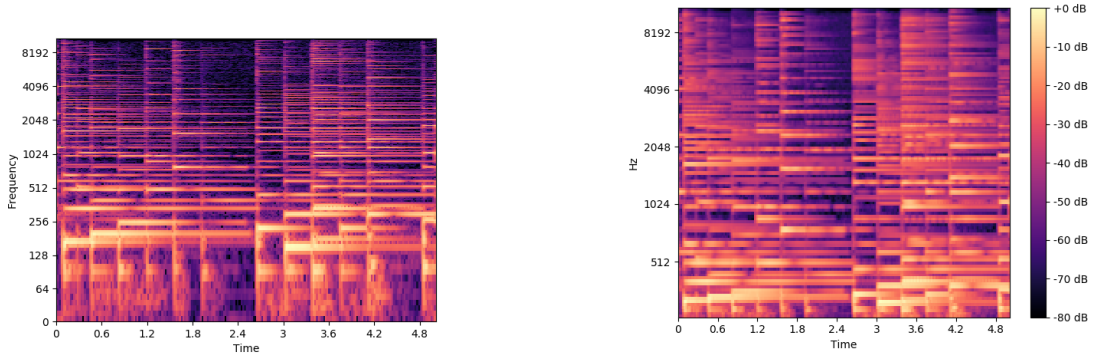
As the first step, filterbank matrix is created according to given audio signal. Librosa.filters.mel function is used to create filterbank matrix by giving sample rate, a number of fft components and maximum frequency of an audio signal. It returns the filterbank matrix(mel transform matrix).

```
filterbank_matrix = librosa.filters.mel( <sample_rate>,
                                         <number_fft_comp>, <max_freq> )
```

After that, mel filter is generated using Numpy.dot function. It returns dot product of given two matrices. Filterbank matrix and STFT matrix are given as the inputs parameters of this function and it returns a mel filter matrix of an audio signal.

```
mel_filter = numpy.dot( filterbank_matrix , stft)
```

Mel scaled matrix for given signal is generated by combining stft bins into mel frequency bins using mel filter and calculate the difference of bins with respect to the filtered spectrogram. Figure 7.5 shows STFT spectrogram view of the signal before and after apply mel filter.




(a) Before apply the mel filter

(b) After apply the mel filter

Figure 7.5: STFT spectrogram view of signal before and after applying the mel filter

After that, the strength of onsets for each time period are extracted from a mel scaled matrix. librosa.onset.onset_strength function is used to extract strengths of onset for given STFT matrix. Aggregation function should be needed to combine onsets in different frequency bins while finding strengths. Therefore median value is used as an aggregation function, because it provides a center value as an aggregated value from given points.

```
onset_strength = librosa.onset.onset_strength(
                    <mel_scaled_matrix>, aggregation = <median> )
```

Peak picking can be identified as the final phase of the onset detection process. It separates onsets and non-onsets from maximum filtered spectrogram. As mentioned in Section 5.2.1 all onsets which fulfil three equations (5.8,5.9,5.10) are selected as peaks of onset. Selected onsets are passed to the periodicity estimation process to estimate the tempo.

As mentioned in Section 5.2.2, autocorrelation method is a good technique for periodicity estimation of soft onsets. Librosa facilitates a direct function for the auto-correlation of a signal.

```
autocorrelation = librosa.autocorrelate(<onset_envelope>)
```

This autocorrelation function provides a truncated data array of an autocorrelation of given onsets. Then tempo estimation is done by analyzing the lag of the larger peaks and using the multiplicity of the relationship among them. Figure 7.6 shows a relationship between original autocorrelated data and shifted version. As mentioned in Section 5.2.2, if there are no relations among peaks in the autocorrelation, tempo estimation only depends on the largest peak.

Librosa possesses a direct function to estimate tempo in an audio signal. But librosa's onset detection function has been implemented using the common spectral flux algorithm.

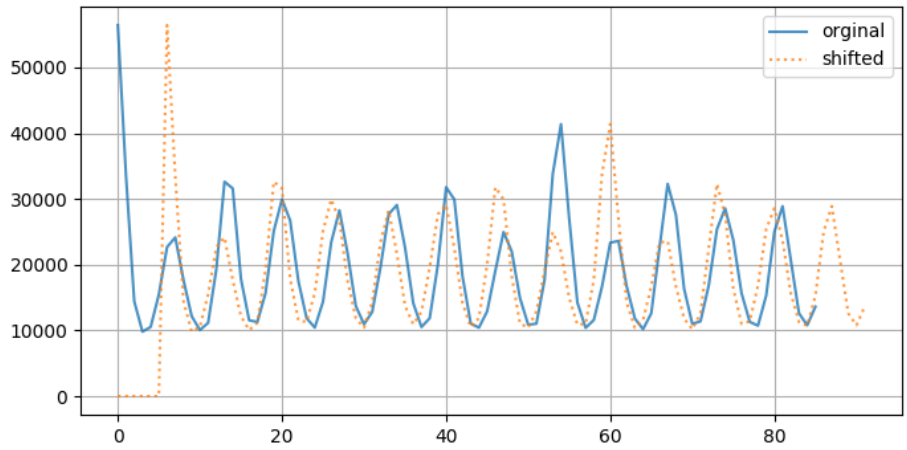



Figure 7.6: Autocorrelated curve of an audio signal and shifted curve of same audio signal

Spectral flux is not a well suitable algorithm for a voice only audio signal. Therefore several modifications were adapted to the librosa function by combining several other methods and by changing the parameters.

Estimated tempo is passed to the next stage for fix the disagreement between estimated tempo and selected tempo. Disagreement fixing process is implemented by using an expression that stated in Section 5.2.3. Although the implementation is light, it holds a large part of this tempo evaluation process because this whole tempo evaluation process is depended on the comparison between selected tempo and estimated tempo. As mentioned in Section 5.2.3, there is a very common problem when human select tempo by tapping along with the music. It may be half of the true tempo or double of the true tempo value.

But this double or half of the tempo value is not an issue for music generating and syncing processes. Therefore double value of the true tempo and half value of the true tempo are also marked as a correct tempo value in the implementation process.

### Chord Generator

Chord generator component is one of the key modules in this system and responsible for decoding phase described in Section 6.3. The python NumPy module was heavily used as there were many matrix operation to be done. The HMM was implemented from scratch as a python class and the same trained data from MySong was used hence the training phase of the HMM was not performed. The python implementation of the Viterbi algorithm appendix C.1 was taken from Wikipedia and adapted it to cater the requirements of this component. Chord Generator component interacts with Pitch Detector component to get the fundamental frequency values of a particular vocal melody and generates an acceptable chord sequence for that melody. The output will be used in accompaniment music generator component to come up with the accompaniment track.

#### Accompaniment Music Generator

The Accompaniment Music Generator component provides the accompaniment track when the chord sequence is presented. The core library to generate this accompaniment music is MMA (see Section 7.1.7). This component is implemented as a python class and the MMA library was integrated to run as a sub process. MMA comes as a command line utility and a file is created with the sequence of chords and specifying rhythm and tempo. This file can be run by the MMA utility to obtain a MIDI file. A MIDI file contains music data, such as what notes are played, when they are played, how long each note is held, and the velocity of each note and it does not contain any sound information. Once a MIDI file is generated, a MIDI player is needed to obtain the actual sound of it. Timidity++ (see Section 7.1.8) was used as the MIDI to WAV converter. Figure 7.7 depicts the composition of the Accompaniment Music Generator

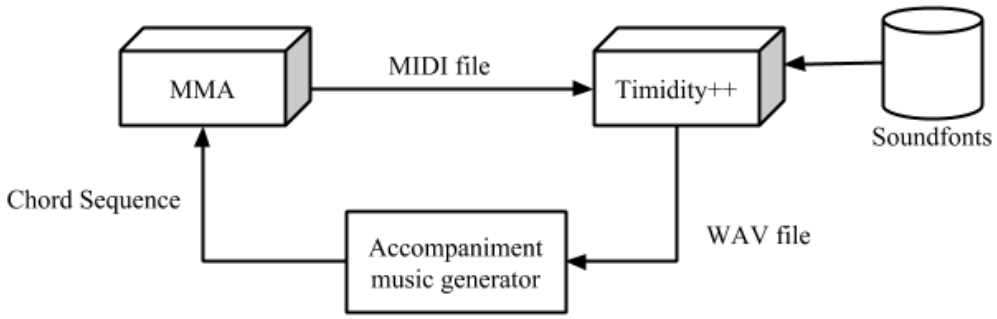



Figure 7.7: The composition of Accompaniment Music Generator

#### Audio Workstation

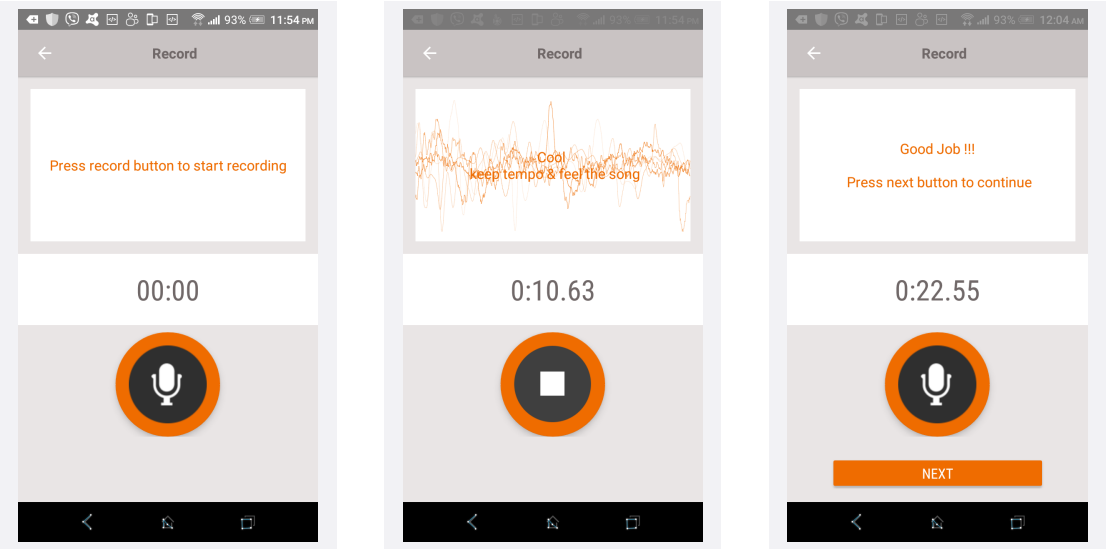
Audio Workstation component is responsible for carrying out the combination of vocal melody with the generated accompaniment music. The python PyDub module (see Section 7.1.10) was used to implement this combination step. Furthermore, several vocal effects for the vocal melody can be applied using this component. The command line utility sox (see Section 7.1.11) was used to generate these effects such as reverb, echo etc. The sox utility was integrated to run as a sub process inside the Audio Workstation Component. Apart from that there is a WAV to MP3 file conversion implemented using LAME (see Section 7.1.9) utility.

## 7.3 Implementation of the Mobile Application

An Android operating system based mobile application was implemented as a client application of this project. Implementation of the mobile application is described using the Android-based technologies under the component of the layered architecture.

### 7.3.1 Presentation Layer

As mentioned in Section 4.2.2, presentation layer was implemented using the inductive interfaces design concepts. XML is used to implement interfaces in native Android application development process. Inductive based implementation is described using record activity of mobile application.




(a) Before the recording (b) During the recording (c) After the recording

Figure 7.8: Screen shots of recording activity

Figure 7.7 shows the screen shots of three states of recording activity. As the first step of inductive concepts, one screen should be focused on one purpose and designer should be introduced this mainly focused area concisely. According to screen shots, record the vocal melody is focused here and it is shown as a title of this activity.

As another step of inductive implementation, there should be a direct instruction to show primary action. According to this implementation process, the primary action of before starting the record is "press the record button to start recording". The primary action during the recording state is "press the stop button to stop the recording". The primary action of after recording state is "press the next button to continue this process". All of these primary actions are stated and highlighted according to the particular state of the activity.

As another step of inductive-based interface designing method, there should be an exit or cancel button to terminate a current process. Termination criteria for each state of this recording activity are also there. This kind of self-explanatory concepts based inductive user interface methods are used to make a correct mental model about particular activities in the mobile application.

### 7.3.2 Business Layer

As mentioned in design Section 4.2.2, high cohesive and loose coupling business logic components are built using less external depended components. An optimistic concurrency controlling mechanism is implemented by sending a time-stamp binded data for transactions without locking resources.

A synchronization problem was identified while implementing the logic of mixing process of recorded vocal melody. The mismatch between recorded audio track and accompaniment music can be identified easily even there exists a 50 milliseconds delay between them. There is an unhandled latency issue [1] of an android advanced audio processing system that caused this problem.Because of that different devices provide different latency values according to device type and current running threads.

Therefore a manual synchronization process is added to the mobile application by changing the logic of automatic music mixing process.

### 7.3.3 Data Layer

Internal audio track management is mainly focused on the implementation of this mobile application data layer. Android default audio management techniques are used to access, transform and manipulate the audio data from the system. By default, meta-data about audio data are stored in SQLite tables.

Data access logic components are implemented by focusing the extraction of meta-data of internal audio tracks. Data helper components are implemented to manipulate data from SQLite tables and add meta-data tags to generated audio tracks.

### 7.3.4 Cross Cutting

Security concerns are implemented under the functionalities of the cross-cutting layer. Vulnerabilities of authentication and authorization strategies are provided security breaches that allow attackers to access sensitive data easily. Therefore, implementation of strong authentication and authorization strategies are mainly focused here. Google APIs based authentication and authorization processes are used to implement the cross-cutting layer functionalities, because of the Android compatible strong security mechanisms of Google APIs and ease of social network integration without additional process overheads.

[1] https://developer.android.com/ndk/guides/audio/audio-latency.html - android audio latency issue

## 7.4 Summary

This chapter presented a detailed description of the tools and technologies, implementation of the server and the mobile application. Service layer of the server handles all the HTTP requests from the mobile application and the language used is Node.js. Core functionalities of the system lies in the business layer of the server and consists of pitch detector, audio workstation, singing skill evaluator, chord generator and accompaniment music generator which were implemented in python 2.7. The mobile application was developed using android based technologies and contains presentation, business and data layers.

# Chapter 8

# Testing and Evaluation

This Chapter elaborates the success rate of the system. The system was evaluated based on the components. Evaluation of pitch based singing skill evaluator and results obtained are described in Section 8.1 and evaluation of the tempo based singing skill evaluator is described in Section 8.2. The evaluation and results of the chord generator component is described in Section 8.3. Finally the description of the usability study is contained in the Section 8.4. The conclusions on the test results are described in the Section 8.5

## 8.1 Test 01 - Pitch Based Singing Skill

This Section outlines the testing methodology and the test result obtained for the key functional requirement of measuring the singing skill mentioned under Section 3.1.2. Classification of singing skill as good or poor based on the pitch of the recorded vocal melody will be tested here.

### 8.1.1 Test Methodology

A sample of 101 vocal melodies was collected from participants including both male and female. All the samples were already existing melodies and length of a sample recording varies between 30 to 60 seconds. 48% of the vocal melodies were recorded from Audacity [1]and the rest was taken from mobile recordings, under the assumption that the recording medium does not affect in singing skill evaluation. As described in Section 5.1, two features were extracted from the collected samples and they had been labeled by an domain expert as good or poor. The Table 8.1 illustrates some of the sample data with the extracted features and the labels given by the expert. Finally the labeled data were classified using an SVM. The performance of the SVM was measured using accuracy, precision and recall values in two steps where cross-validation was absent and the cross-validation was present. 70% of

[1]'https://www.audacityteam.org/'

the sample recordings were selected to construct the training set for the SVM and 30% of the sample recordings were selected to construct the testing set.

Table 8.1: Sample data with the extracted features and the labels given by the expert

| **Song ID** | **feature 1** | **feature 2** | **label** |
|---|---|---|---|
| 15092017020510.wav | 564763.385 | -0.0981 | good |
| 15092017021603.wav | 616612.955 | -0.0487 | good |
| 22122017123454.wav | 655493.486 | -0.011 | poor |
| 22122017124258.wav | 658866.654 | -0.008 | poor |

### 8.1.2 Test Results

Table 8.2 presents the results obtained for the performance of the SVM when the cross-validation was absent and a value of 0.77 was obtained for each of the accuracy, precision and recall value. It can be seen from the data in Table 8.3 that the values of accuracy, precision, recall have obtained 0.818, 0.763 and 0.878 respectively when the k-fold cross validation was present with 5 folds.

Table 8.2: Performance of the SVM classifier without cross-validation

| **Measure** | **Value** |
|---|---|
| accuracy | 0.77 |
| precision | 0.77 |
| recall | 0.77 |

Table 8.3: Performance of the SVM classifier with 5-fold cross-validation

| **Measure** | **Value** |
|---|---|
| accuracy | 0.818 |
| precision | 0.763 |
| recall | 0.878 |

## 8.2 Test 02 - Tempo Based Singing Skill

This Section presents testing methodology and result of tempo based singing skill evaluation of audio recordings in order to assess the functional requirement which is stated under Section 3.1.2. The tempo estimation algorithm and the classification of singing skill as good or poor based on the tempo of the recorded vocal melody will be tested here.

### 8.2.1 Test Methodology

Testing of this model is divided into two main parts. The test methodology of the tempo estimation algorithm is presented at first and the test methodology of the tempo based singing classifier is presented at second.

#### Tempo Estimation Algorithm

The database used to evaluate this tempo based singing skill evaluation module is constituted of 60 short segments of musical audio signals (each of 30 seconds long). 42% of the audio recordings were Acapella : A kind of instrumental music that generated using only percussion instruments. 26% of the audio recordings were mobile recordings of only the vocal melodies that were recorded using default recording application in mobile devices. 12% of the audio recordings were complete songs where each song contains both the vocal melody and the accompaniment music. 8% of the audio recordings were instrumental music where both harmonic and percussive instruments were present. 7% of the audio recordings were harmonic instrumental music whereas the rest of the audio recordings were percussion instrumental music.

The accuracy of the tempo estimation algorithm is tested by comparing the outcome of the algorithm against the actual tempo of the audio recordings manually annotated by musicians.

#### Tempo Based Singing Skill Classifier

A sample of 24 vocal recordings was collected using an implemented mobile application. The tempo of these samples has been manually estimated and the overall tempo based singing skill was evaluated by an expert and they were labeled as good or poor. The Table 8.1 shows some of the sample data (see appendix B.2 for complete table) with the intended tempo by the user, the estimation of the tempo by the system, the labels given by the expert and the classes selected by the system. Finally the performance of the tempo based singing skill classifier was measured using accuracy, precision and recall values.

Table 8.4: Sample data with the estimated tempo values and the labels given by the expert

| **Song ID** | **User selected tempo** | **System tempo estimation** | **Expert's tempo evaluation** | **System tempo evaluation** |
|---|---|---|---|---|
| Song 1 | 110 | 112 | Good | Good |
| Song 2 | 90 | 92 | Good | Good |
| Song 3 | 94 | 103 | Poor | Poor |

### 8.2.2 Test Results

Test results of tempo estimation algorithm and tempo based singing skill classifier are described here.

#### Tempo Estimation Algorithm

As can be seen from the Table 8.5, 72% of all the audio recordings were estimated as equal to the actual tempo of the songs, 18% were estimated as equal to a half of the actual tempo of the songs, 8% were estimated as equal to a double of the actual tempo of the songs and the rest were estimated incorrectly.

Table 8.5: Evaluation result of the tempo estimation algorithm

| **Song categories** | **Number of songs** | **Songs with actual tempo** | **Songs with half of actual tempo** | **Songs with double of actual tempo** | **Wrong estimations** |
|---|---|---|---|---|---|
| Acapella | 25 | 21 | 0 | 4 | 0 |
| Percussion Instrumental | 3 | 3 | 0 | 0 | 0 |
| Harmonic Instrumental | 4 | 2 | 2 | 0 | 0 |
| Instrumental | 5 | 3 | 2 | 0 | 0 |
| Complete Songs | 7 | 5 | 1 | 1 | 0 |
| Mobile Recordings | 16 | 9 | 6 | 0 | 1 |
| **All** | 60 | 43(72%) | 11(18%) | 5(8%) | 1(2%) |

#### Tempo Based Singing Skill Classifier

It can be seen from the data in Table 8.6 that the values of accuracy, precision, recall have obtained 0.875, 0.944 and 0.894 respectively as the performance for the tempo based singing skill classifier.

Table 8.6: Performance of the tempo based singing skill classifier

| **Measure** | **Value** |
|---|---|
| accuracy | 0.875 |
| precision | 0.944 |
| recall | 0.894 |

## 8.3 Test 03 - Chord Generation

As described in Section 3.1.3, generation of accompaniment music for a vocal melody is a key requirement of this system. The accompaniment music is solely based on the selected chords for a particular melody. Testing of the chord generation component is described in detail in this section.

### 8.3.1 Test Methodology

Chord generator component is based on a Machine Translation (MT) model and MT metrics can be applied to evaluate the chord generator component's accuracy. One of the common MT metric is the Word Error Rate ($WER$) which is used as a performance measure of speech recognition or machine translation systems. The $WER$ metric is a valuable tool for comparing different systems and in this section the output of the chord generation component is compared with an output from an domain expert. $WER$ and the $WAcc$ (Word Accuracy) are defined using the terminology of this project in the equation 8.1 and equation 8.2 respectively.

$$WER = \frac{number\ of\ incorrectly\ identified\ chords}{number\ of\ all\ chords\ identified} \tag{8.1}$$

$$WAcc = 1 - WER \tag{8.2}$$

A sample of 14 vocal melodies were recorded, all by the same vocalist ranging from twenty to thirty seconds in length. All melodies were in a major key and they were from independent artists. An original chord progressions for all the melodies were prepared by a music expert and let this be referred to as 'Original Reference'. The output from the chord generator for all of these melodies were combined and let this be referred to as a 'Candidate'. As described in the Section 6.3.5 "Happy Factor" defines mode of the song: major or minor. The most suitable Happy Factor value was selected using a trial and error approach by choosing the Happy Factor value for which the $WER$ of the "Candidate" compared to "Original Reference" was at its minimum. Let us name this candidate as the "Optimal Candidate". This Optimal Candidate was refined by the same expert and let this be referred to as the "Refined Reference". Finally, the $WER$ for the "Optimal Candidate" was calculated against the "Refined Reference" to get the overall Word Error Rate for the chord generator component. The refinement from the "Optimal Candidate" to "Refined Reference" was reasonable because of the goal of automatic harmonization of vocal melodies is not to obtain the correct chord sequence but to extract more appropriate chord sequence from the melody.

### 8.3.2 Test Results

Table 8.7 presents the relevant results obtained from the trial and error method (see appendix B.1 for complete table) to find out the suitable happy factor value. It can be seen from the data that the $WER$ become minimum when Happy Factor value equals to 0.55. Table 8.8 provides the $WER$ of each song when the "Happy Factor" equals to 0.55 and the overall $WER$ of 10.619% has been achieved by the chord generator.

Table 8.7: $WER$ when compared with "Original Reference" for different "Happy Factor" values

| **Happy Factor** | **Word Error Rate (Original Reference)** |
|---|---|
| 0.0 | 77.434% |
| 0.54 | 36.190% |
| 0.55 | 35.841% |
| 0.56 | 38.496% |
| 1.0 | 57.965% |

Table 8.8: $WER$ when "Happy Factor" is 0.55 and compared with "Refined Reference"

| **Song ID** | **Number of Measures** | **Wrong Chords** | **Word Error Rate** |
|---|---|---|---|
| Song 1 | 16 | 3 | 18.8% |
| Song 2 | 20 | 2 | 10.0% |
| Song 3 | 20 | 0 | 0.0% |
| Song 4 | 16 | 2 | 12.5% |
| Song 5 | 16 | 1 | 6.2% |
| Song 6 | 16 | 1 | 6.2% |
| Song 7 | 16 | 3 | 18.8% |
| Song 8 | 16 | 1 | 6.2% |
| Song 9 | 16 | 0 | 0% |
| Song 10 | 8 | 2 | 25% |
| Song 11 | 16 | 2 | 12.5% |
| Song 12 | 16 | 4 | 25% |
| Song 13 | 18 | 1 | 5.6% |
| Song 14 | 16 | 2 | 12.5% |
| **Total** | **226** | **24** | **10.619%** |

## 8.4 Test 04 - Usability Test

The usability test was carried out to evaluate the achievement of our research aim by testing the overall system on users.

### 8.4.1 Test Methodology

17 participants including both male and female were selected for the usability test. A brief introduction about our system was given to each participant and asked to come up with a reasonable accompaniment music for their recorded vocal melody. Each one of the participants were given 10 minutes to complete this task. Once the task was completed, the users were provided with 6 Likert-scale questions of five-levels to obtain the feedback about the usability of our system. Each of the five-levels: strongly disagree, disagree, neutral, agree and strongly agree were assigned a weight from 1 to 5 respectively in the increasing order. A conclusion for the usability of our application can be derived based on the mean values calculated using the assigned weights for all the responses provided to each question.

### 8.4.2 Test Results

Table 8.9 presents the six questions and the mean values obtained for them(see appendix B.3 for complete set of responses). The overall responses for each questions were positive.

Table 8.9: Summary of the Likert-scale questionnaire

| Questions | Mean Value |
|---|---|
| I enjoyed using this system | 4.53 |
| I Gained a good understanding about this system | 4.53 |
| I felt I was able to create reasonable music using this system | 3.76 |
| Singing skill evaluation encouraged me to sing better | 4.00 |
| Synchronization of tracks was easy | 3.76 |
| I would use this system for entertainment if I had this software | 4.58 |

## 8.5 Discussion

The results of the Test 01 indicate that the pitch based singing skill evaluator component can classify the singing skill of a user based on pitch reasonably well with 77% of accuracy when the cross-validation was absent in the training process. The classifier accuracy has increased to 81.8% when the 5-fold cross-validation was present in the training process. The reasons for the slight error may be because of the background noise, vibrato and sudden variations of some vocal melodies.

The results of the Test 02 shows that the tempo estimation algorithm can perform really well and one unanticipated result was that there was an incorrect tempo estimation from a mobile recording. A possible explanation for this might be the extreme and sudden variations in pitch of that vocal recording which generates larger peaks of random onsets. Another important result from Test 02 was that the tempo based singing skill evaluator was able to classify the singing skill based on the tempo really well at the accuracy of 87.5%.

The results of test 03 was successful as the chord generator component was able to select appropriate chords for the vocal melody with a $WER$ (Chord Error Rate) of 10.619%. In other terms, the $WAcc$ (Chord Accuracy) of the chord generator was 89.381%. We hope that the minor mode songs would generate similar results for the same testing procedure. Another important finding from Test 03 is that the chord generator was able to identify the original key of each of the 14 vocal melodies successfully.

We are particularly enthusiastic about the results of the Test 04 as the mean response for all the questions were above 3.76. The majority of the respondents felt that our system can be used for entertainment and they were interested in continued use of our system.

## 8.6 Summary

This chapter described the testing and evaluation of main components and the usability of the system. Pitch based singing skill evaluation component used an SVM as the binary classifier to classify vocal melodies as good/poor, and the model performed well with a higher accuracy, precision and recall. Tempo based singing skill evaluation component was tested under two categories: testing of tempo estimation algorithm and testing of tempo based singing classifier. Accuracy of tempo estimation algorithm was evaluated by comparing the outcome of the algorithm against the actual tempo of the audio recordings. Tempo based singing classifier was tested by comparing the system generated tempo classification against a domain expert's tempo classification as good/poor. The classifier performed really well with a higher accuracy. $WER$ (Word Error Rate), an MT metric was used to evaluate chord generation component. The output of the chord generation component is compared with the output from a domain expert and reasonable results were obtained.

# Chapter 9

# Conclusion

This chapter discusses the conclusions about the project. Section 9.1, Section 9.2 and Section 9.3 present the conclusions which can be arrived at the project aim and objectives, limitations that can be seen in the current project and the implications for the further research respectively.

## 9.1 Conclusion on Project Aim and Objectives

This project was undertaken to design and develop a mobile application which can generate accompaniment music when a user sings a song. The main aim of this project is to allow non-musicians to get a taste of music composition. Classifying the singing skill of the user based on pitch and tempo was the first objective to achieve the specified aim. The purpose of this objective was to encourage the user to improve their singing skill to obtain more pleasing accompaniment music from the system. The second objective was to select an acceptable chord sequence for a vocal melody to generate the accompaniment music.

The proposed approach for the singing skill evaluation consisted of two steps. As the first step, a binary classifier for singing skill of the user was implemented based on the pitch interval accuracy of the vocal melody and the classifier performed reasonably well with an accuracy of 81.8%. As the second step, a binary classifier for singing skill of the user was implemented based on the tempo of the vocal melody. A vibrato suppression based onset detection technique was used in the tempo estimation algorithm and the classifier was able to perform really well with an accuracy of 87.5%. The minor error in the accuracies of the classifiers may be because of the noise, vibrato and sudden variations in the vocal melodies. These findings suggest that the implemented system was able to identify the singing skill of the user in general as good/poor based on pitch and tempo of the vocal melody as expected.

The generation of accompaniment music is solely based on the automatic harmonization of vocal melodies. The automatic harmonization was achieved using a Hidden Markov Model and the decoding step of the model was done using the Viterbi algorithm to come up with the best possible sequence of chords for the vocal melody. The automatic harmonization

method was able to select chords at a rate of 89.381% accuracy.

Finally, we conclude that our system is capable of generating acceptable accompaniment music for vocal melodies.

## 9.2 Limitations

Several limitations to this system need to be acknowledged. First, the pitch based singing skill evaluator is only able to perform a binary classification because of the lack of training data. The second limitation of the system is that the variation of chord kinds are very few as only 5 chord kinds were selected when training the model. The third and the last limitation is the exact determination of the "Happy Factor" value when generating chords for a melody.

## 9.3 Implications for Further Research

It is recommended that further research be undertaken in the following areas: increasing the classes of the pitch based singing skill classifier, removing the "Happy Factor" and improving the accuracy of the chord generation process. In a personal interview with Mr. Suresh Maliyadda, he suggested that the "Happy Factor" does not really define the feel of the song correctly. We are thus excited to explore on classification of vocal melodies based on the mode:major or minor. Furthermore, recent techniques of sequence to sequence models such as Long Short Term Memory architectures can be used to improve the chord generation process.

# References


[1] I. Simon, D. Morris, and S. Basu, "MySong: automatic accompaniment generation for vocal melodies," *Proceedings of ACM CHI 2008 Conference on Human Factors in Computing Systems*, vol. 1, pp. 725–734, 2008.

[2] D. Morris, I. Simon, and S. Basu, "Exposing parameters of a trained dynamic model for interactive music creation," *Proc AAAI 2008*, no. 2003, pp. 784–791, 2008.

[3] T. Fujishima, "Realtime Chord Recognition of Musical Sound: A System Using Common Lisp Music," pp. 464–467, 1999.

[4] C.-H. Chuan and E. Chew, "A Hybrid System for Automatic Generation of Style-Specific Accompaniment," *Proceedings of the 4th International Joint Workshop on Computational Creativity*, pp. 57–64, 2007.

[5] P.-p. Singing, "P -p s," pp. 95–115, 2007.

[6] M. Mauch, K. Frieler, and S. Dixon, "Intonation in Unaccompanied Singing : Accuracy , Drift and a Model of Reference Pitch Memory," *The Journal of the Acoustical Society of America*, vol. 136, no. 1, pp. 1–11, 2014.

[7] P. ŻWAN, "Automatic singing quality recognition employing artificial neural networks," *Archives of Acoustics*, vol. 33, no. 1, pp. 65–71, 2008.

[8] T. Nakano and M. Goto, "An Automatic Singing Skill Evaluation Method for Unknown Melodies Using Pitch Interval Accuracy and Vibrato Features," pp. 1706–1709, 2006.

[9] M. Alonso, B. David, and G. Richard, "Tempo and beat estimation of musical signals," *Proc International Conference on Music Information Retrieval*, vol. 04, pp. 158–163, 2004.

[10] T. R. Agus, C. Suied, S. J. Thorpe, D. Pressnitzer, T. R. Agus, S. J. Thorpe, and D. Pressnitzer, "To cite this version : Characteristics of human voice processing," no. May 2010, pp. 509–512, 2011.

[11] C. C. Toh, B. Zhang, and Y. Wang, "Multiple-Feature Fusion based Onset Detection for Solo Singing Voice," *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR)*, pp. 515—-520, 2008.

[12] D. A. Effects, "MAXIMUM FILTER VIBRATO SUPPRESSION FOR ONSET DETECTION Sebastian Böck and Gerhard Widmer Department of Computational Perception Johannes Kepler University," pp. 1–7, 2013.

[13] P. Boersma, "Accurate Short-Term Analysis of the Fundamental Frequency and the Harmonics-To-Noise Ratio of a Sampled Sound," *Proceedings of the Institute of Phonetic Sciences*, vol. 17, pp. 97–110, 1993.

[14] Microsoft, "Mobile Application Architecture Guide," p. 138, 2008.

[15] "Systems and software engineering – vocabulary," *ISO/IEC/IEEE 24765:2010(E)*, pp. 1–418, Dec 2010.

[16] D. Gerhard, "Pitch extraction and fundamental frequency: History and current techniques," *Time*, pp. 0–22, 2003.

[17] B. Jannsen, "Support Vector Machines for Binary Classification and its Applications," pp. 0–65, 2008.

[18] S. Okamura, "The Short Time Fourier Transform and Local Signals," 2011.

[19] B. Logan, "Mel Frequency Cepstral Coefficients for Music Modeling," *International Symposium on Music Information Retrieval*, vol. 28, p. 11p., 2000.

[20] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, Feb 1989.

[21] G. D. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, March 1973.

# Appendices

# Appendix A

# Contribution

## A.1 Individual Contribution for the Project

Each member contributed to the project work nearly equally in various ways.

Pitch based singing skill evaluator was developed by Ms.K.A.K.Indrachapa. A background study on the singing skill evaluation of unknowns melodies was carried out. A task of noise filtering from the vocal melodies was also done. features related to pitch interval accuracy were extracted and trained a classifier with the extracted features.The evaluation of the singing skill classifier was conducted. Understanding the complex equations and implementing them were a challenging tasks. Improving the efficiency and performance of the skill evaluator is another difficult task.

The implementation of the pitch detector, service layer using node.js and server prototype was carried out by Mr.D.H.U.Perera. Furthermore, the usability test was conducted. A Praat Script was written to extract the fundamental frequency values from the vocal melody. Understanding the function of Praat and adapt it to cater our requirement was challenging. Node.js 6 was new and learning it was quite challenging. Additional to that the trained data was adapted as required by the system to use in the Hidden Markov Model. Working with these datasets with lack of music knowledge was another challenging task. Mr.D.H.U.Perera was helped by the group to overcome these problems, specially the music barrier and also relevant research papers was very useful as well.

Chord generation and accompaniment music generation components were developed by Mr.R.W.M.N.H.Wanigasekera. A background study on automatic harmonization of vocal melodies was done first and identified the techniques. Further analysis on LSTMs was carried out to assess the suitability of it to our project. Domain knowledge for this project was provided. The overall architecture for the web server was designed and the composition of the software components was also done. The evaluation of chord generation process was also carried out. Understanding the LSTMs and HMM was a difficult task. Furthermore, adapting the Viterbi algorithm and understanding its process is also a challenging task.

Tempo-based singing skill evaluator and main functionalities of the mobile application

were developed by Mr.W.K.P.Wanniachchi. A background study on the tempo based singing skill evaluation was performed priori to this research. Even though there were many tempo evaluation methods exists, a directly applicable methods for the tempo evaluation of voice only melodies were not there. According to the priori studies, it was clear that there were many tempo estimation algorithms which used the percussion onsets for tempo estimation. The dynamic behavior of voice impacted a lot for the less availability of the direct applicable algorithms for tempo detection in human voice. A novel approach in detecting the tempo in human voice had been developed. Advanced audio processing of Android mobile application was difficult because of the Android audio latency. Manual synchronization process was added to handle that latency issue. Interface design for various type of devices, android life cycle handling and memory management were also done. Evaluation of the tempo based singing skill classifier was also carried out.

Finally, 95% completion of the project was achieved.

# Appendix B

# Tables and Diagrams

## B.1 Result of the Chord Generation Test

Table B.1: Chord generation test results according to happy factor values

| **Happy Factor** | **Word Error Rate (Original Reference)** | **Word Error Rate (Refined Reference)** |
|---|---|---|
| 0.0 | 77.434% | 71.239% |
| 0.30 | 51.770% | 37.168% |
| 0.40 | 46.903% | 32.743% |
| 0.50 | 40.265% | 21.239% |
| 0.53 | 37.168% | 15.044% |
| 0.54 | 36.190% | 11.905% |
| 0.55 | 35.841% | 10.619% |
| 0.56 | 38.496% | 15.487% |
| 0.57 | 39.381% | 18.584% |
| 0.60 | 43.363% | 24.336% |
| 0.70 | 48.230% | 35.398% |
| 1.0 | 57.965% | 52.212% |

## B.2 Tempo Based Singing Skill Evaluator Test Data

Table B.2: Sample data with the estimated tempo values and the labels given by the expert

| **Song ID** | **User selected tempo** | **System tempo estimation** | **Expert's tempo evaluation** | **System tempo evaluation** |
|---|---|---|---|---|
| Song 4 | 102 | 103 | Good | Good |
| Song 5 | 100 | 129 | Poor | Poor |
| Song 6 | 106 | 107 | Good | Good |
| Song 7 | 100 | 103 | Good | Good |
| Song 8 | 95 | 117 | Poor | Poor |
| Song 9 | 102 | 103 | Good | Good |
| Song 10 | 110 | 112 | Good | Good |
| Song 11 | 100 | 103 | Good | Good |
| Song 12 | 108 | 117 | Good | Poor |
| Song 13 | 92 | 92 | Good | Poor |
| Song 14 | 102 | 107 | Poor | Good |
| Song 15 | 120 | 123 | Good | Good |
| Song 16 | 88 | 107 | Good | Poor |
| Song 17 | 132 | 129 | Good | Good |
| Song 18 | 60 | 58 | Good | Good |
| Song 19 | 95 | 95 | Good | Good |
| Song 20 | 125 | 129 | Good | Good |
| Song 21 | 110 | 107 | Good | Good |
| Song 22 | 95 | 95 | Good | Good |
| Song 23 | 100 | 107 | Poor | Poor |
| Song 24 | 100 | 99 | Good | Good |

## B.3 Usability Test Statistical Data

Table B.3: Results of usability test

| **Questions** | **Strongly Disagree** | **Disagree** | **Neutral** | **Agree** | **Strongly Agree** |
|---|---|---|---|---|---|
| I enjoyed using this system | 0 | 0 | 1 | 6 | 10 |
| I Gained a good understanding about this system | 0 | 0 | 2 | 4 | 11 |
| I felt I was able to create reasonable music using this system | 0 | 0 | 5 | 11 | 1 |
| Singing skill evaluation encouraged me to sing better | 0 | 0 | 6 | 5 | 6 |
| Synchronization of tracks was easy | 0 | 3 | 4 | 4 | 6 |
| I would use this system for entertainment if I had this software | 0 | 0 | 0 | 7 | 10 |

## B.4 Usecase Diagram of the System

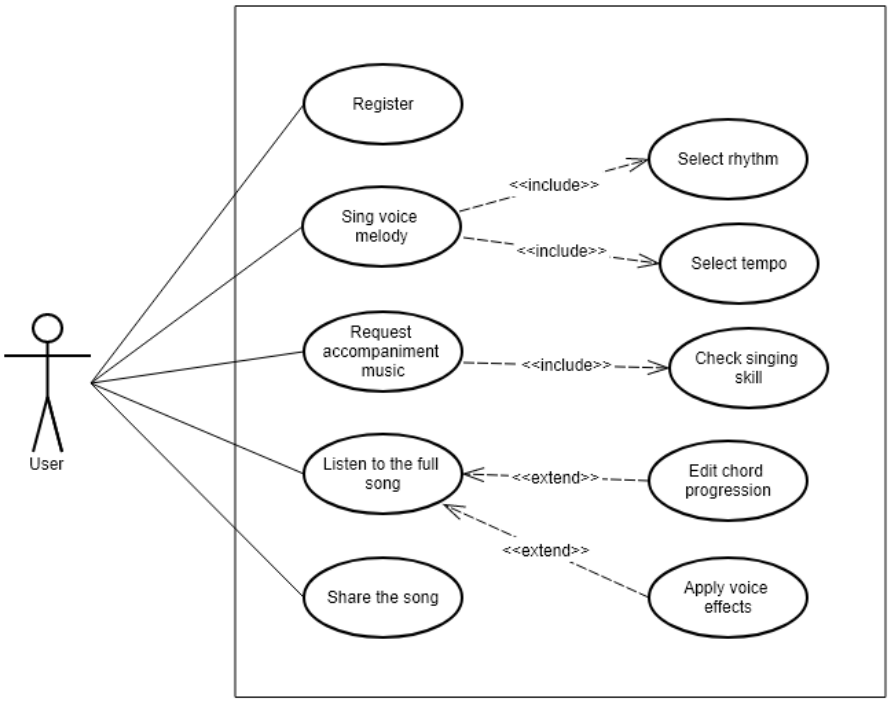



Figure B.1: Usecase diagram of the system

## B.5 Activity diagram of a user creating a new song

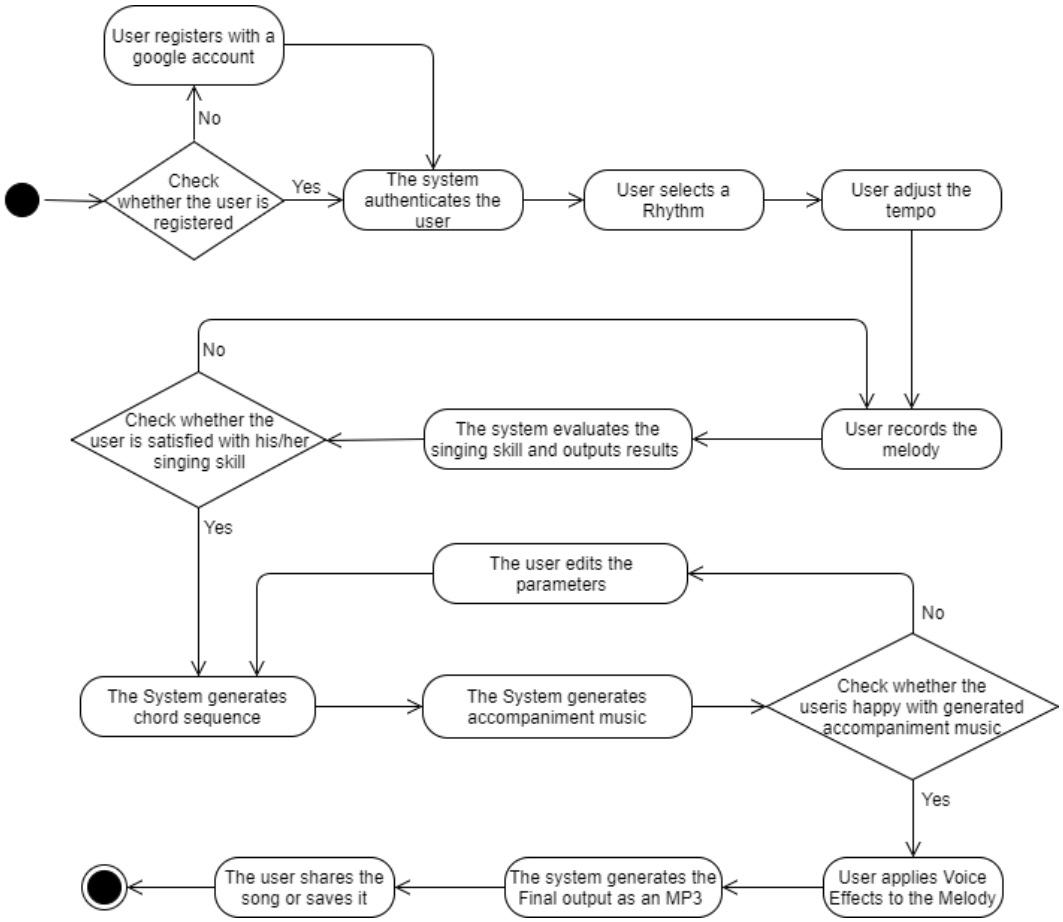



Figure B.2: Activity diagram of a user creating a new song

# Appendix C

# Code Listings

## C.1 Python Implementation of Viterbi Algorithm

```
def viterbi(self,obs, states, start_p,
        trans_p, emit_p,chord_index):
        V = [{}]
        for st in states:
            V[0][st] = {"prob": start_p[chord_index.get(st)]
            * emit_p[chord_index.get(st)][obs[0]], "prev": None}
        # Run Viterbi when t > 0
        for t in range(1, len(obs)):
            V.append({})
            for st in states:
                max_tr_prob = max(
                    V[t - 1][prev_st]["prob"]
                    * trans_p[chord_index.get(prev_st)]
                    [chord_index.get(st)]
                    for prev_st in states)
                for prev_st in states:
                    if V[t - 1][prev_st]["prob"]
                    * trans_p
                    [chord_index.get(prev_st)]
                    [chord_index.get(st)]
                    == max_tr_prob:
                        max_prob = max_tr_prob
                        * emit_p[chord_index.get(st)][obs[t]]
                        V[t][st]
                        = {"prob": max_prob, "prev": prev_st}
```

```
26                          break
27          opt = []
28          # The highest probability
29          max_prob = max(value["prob"] for value in V[-1].values())
30          previous = None
31          # Get most probable state and its backtrack
32          for st, data in V[-1].items():
33              if data["prob"] == max_prob:
34                  opt.append(st)
35                  previous = st
36                  break
37          # Follow the backtrack till the first observation
38          for t in range(len(V) - 2, -1, -1):
39              opt.insert(0, V[t + 1][previous]["prev"])
40              previous = V[t + 1][previous]["prev"]
41
42          return max_prob, opt
```

## C.2 Praat Script for Pitch Detection

```
1  form Variables
2      sentence filename
3  endform
4  Read from file... 'filename$'
5  To Pitch (ac)... 0.0 75.0 15 off 0.03 0.45 0.01 0.35 0.14 600.0
6  frames = Get number of frames
7  output$ = "Time"+tab$+"Pitch"+newline$
8  for f from 1 to frames
9      t = Get time from frame number... 'f'
10     t$ = fixed$(t, 3)
11     v = Get value in frame... 'f' Hertz
12     v$ = fixed$(v, 2)
13     output$ = output$+t$+tab$+v$+newline$
14 endfor
15 echo 'output$'
```

## C.3 Function for M value computation

```
def calculate_M(grid_frequency,output,ltm):
    mi = grid_frequency - 50
    ma = grid_frequency+ 50
    for F in range(mi, ma):
        F2=0
        if F < 0:
            F2 = 100 + F
        if F >= 100:
            F2 = F - 100
        else:
            F2 = F
        output.append( (grid_frequency - F) ** 2 * ltm[F2])
    M=simps(output,range(mi,ma))
        return M
```

## C.4 slope value computation

```
def compute_GF(grid_frequency, ltm):
    GF_val = [0 for i in range(50)]
    for f in range(0, 50):

        plus = grid_frequency + f
        minus = grid_frequency - f

        if (plus >= 100):
            plus = plus - 100
        if (minus < 0):
            minus = minus + 100
        avg = (ltm[plus] + ltm[minus]) / 2.0
        GF_val[f] = avg
    return GF_val

y = compute_GF(grid_frequency=Fg, ltm=gf)
x = np.arange(0, len(y))
slope, intercept, r_value, p_value, std_err = stats.linregress(x, y)
```