AUDIO TO MIDI CONVERTER WITH SYNTHESIZER

PROJECT REPORT

Submitted in partial fulfilment of the
Requirements for the degree of

BACHELOR OF ENGINEERING
In
COMPUTER ENGINEERING

By

Seat No.	Group No. 10
________________	Murtaza Dhuliawala
________________	Sanjay Khatwani
________________	Hirak Modi

Under the guidance of

Dr. Archana B. Patankar

(Associate Professor, Computer Engineering Department)

Computer Engineering Department
Thadomal Shahani Engineering College
University of Mumbai
2013-2014

Project Entitled: **AUDIO TO MIDI CONVERTER WITH SYNTHESIZER**

Submitted By: Murtaza Dhuliawala

Sanjay Khatwani

Hirak Modi

in partial fulfillment of the degree of B.E. in Computer Engineering is approved.

Guide	Examiners
______________________	___________________
Dr. Archana B. Patankar (Associate Professor)	
	___________________
______________________	___________________
Mr. Jayant Gadge (Head of the Department)	Principal

Date: ________________

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

ADSR – Attack Decay Sustain Release

AMT – Automatic Music Transcription

DAW – Digital Audio Workstation

DFT – Discrete Fourier Transform

FFT – Fast Fourier Transform

MIDI – Musical Instrument Digital Interface

Music IR – Music Information Retrieval

NAE – Note Average Energy

SNR – Signal to Noise Ratio

VST – Virtual Studio Technology

1. INTRODUCTION

This chapter gives a brief introduction to the project. It explains the motivation for selecting the project by pointing out the various shortcomings in existing systems. The extent to which the developed system works is also explained in this chapter along with an organization of the report.

1.1. Abstract

Music transcription is a complex cognitive task that requires a trained musician to listen to a piece of music, write down what notes were played and the timing of the notes. The task is further complicated if the music is polyphonic, where several notes are played simultaneously, requiring the musician to listen repeatedly to the piece of music so as to work out the notes that were played and their timing.

Audio to MIDI (Musical Instrument Digital Interface) converter with synthesizer is a great tool for musicians and aims to accomplish the above mentioned transcription. Not only transcribing but also converting it into the digital format. It allows users to play any musical instrument without actually having to learn how to play it. Audio is an electrical or other representation of sound, while MIDI carries event messages that specify pitch and velocity that are sent to other devices where they control sound generation and other features. An acoustic instrument produces audio waves which are given to Audio to MIDI converter which converts the input to a MIDI file. The main advantage of MIDI file is that it is noise free and its size is very small. The MIDI file is given to a synthesizer which produces the sound of the desired tone independent of the acoustic instrument on which the music piece was played originally. Also the MIDI file could be given to the module that generates sound from soundfonts giving the effect of any instrument that the user intends it to be, irrespective of the instrument on which a particular sound piece was originally played. This makes it really easy for composers to put sounds from various instruments without having to actually learn to play them.

1.2. Introduction and Motivation

1.2.1. Introduction

With the increasing interest in music and related technologies these days the importance of Music Information Retrieval and concise representations in which audio tracks can be stored, like MIDI format are gaining popularity. A lot of research is now going into developing methods and techniques in this field that can make it possible to do things which one couldn't even think of earlier.

An audio signal is a representation of sound, typically as an electrical voltage. Audio signals have frequencies in the audio frequency range of roughly 20 to 20,000 Hz (the limits of human hearing). Audio signals may be synthesized directly, or may originate at a transducer such as a microphone, musical instrument pickup, phonograph cartridge, or tape head. Loudspeakers or headphones convert an electrical audio signal into sound. Figure 1-1 shows the time domain representation of an audio signal.

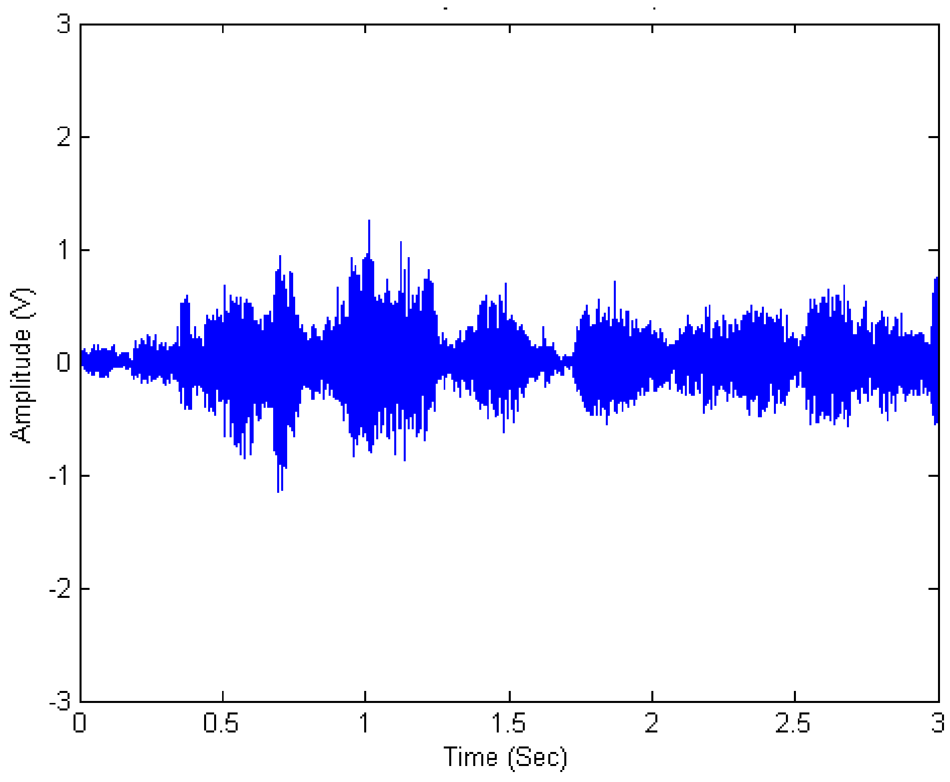


Figure 1-1: Audio Signal

MIDI is a technical standard that describes a protocol, digital interface and connectors and allows a wide variety of electronic musical instruments, computers and other related devices to connect and communicate with one another. A single MIDI link can carry up to sixteen channels of information, each of which can be routed to a separate device.

MIDI carries event messages that specify notation, pitch and velocity, control signals for parameters such as volume, vibrato, audio panning, cues, and clock signals that set and synchronize tempo between multiple devices. These messages are sent to other devices where they control sound generation and other features. This data can also be recorded into a hardware or software device called a sequencer, which can be used to edit the data and to play it back at a later time. Figure 1-2 shows the Piano Roll representation of MIDI file.

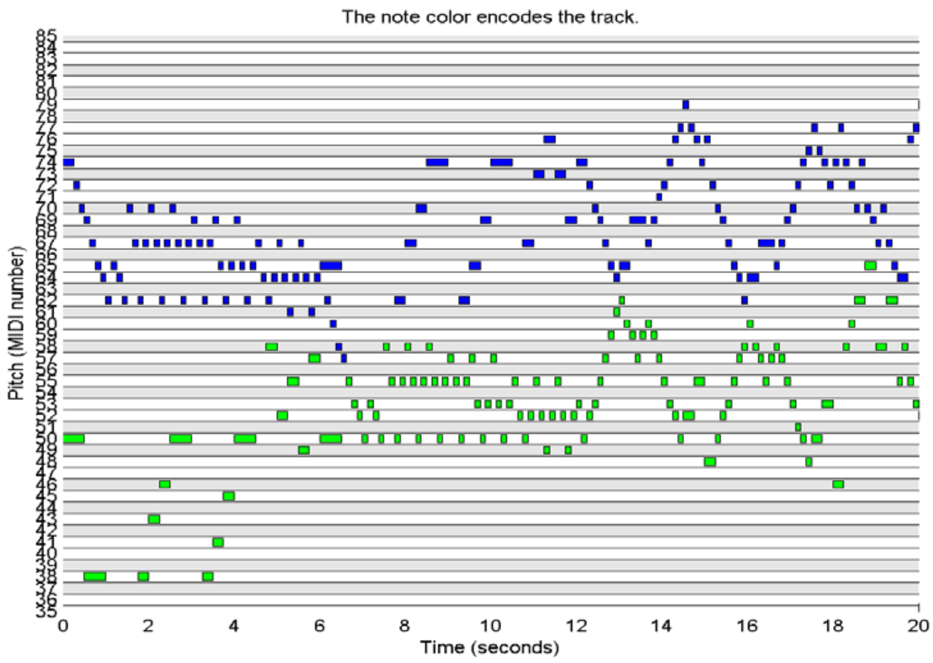


Figure 1-2: Piano Roll

Advantages of MIDI are, file sizes are smaller, all aspects of the sound can be edited and effects can be applied to individual instruments, no interference or background noise during recording and can be used to create music without need for different musicians / instruments.

A sound synthesizer (often abbreviated as “synthesizer” or “synth”) is an electronic instrument capable of producing a wide range of sounds. Synthesizers may either imitate other instruments (imitative synthesis) or generate new timbres. They can be played (controlled) via a variety of different input devices (including keyboards, music sequencers and instrument controllers). Synthesizers generate electric signals (waveforms), and can finally be converted to sound through loudspeakers or headphones.

1.2.2. Motivation

The motivation behind this project has been to provide musicians with a tool that would reduce the overhead of learning more and more instruments or searching for artists who could play for them in the making of any composition.

The objective is to give composers and musicians more time to be creative and artistic. It would enable them to focus more on the composition itself rather than the instruments that they wish to put in it.

What this signifies is that a guitarist can play notes on the guitar and get the sound of a piano as a part of his or her composition even though he or she wouldn’t know how to play it. So it will save the time and efforts of composer in learning piano and he or she can compose whole song on guitar only. Above is just a small example, but the sound of various other difficult to learn instruments or rare instruments could be used by composers using an application like this one.

This flexibility that a system of this sort would provide the musicians has been the biggest motivation for the project team.

1.3. Problem Statement

In this project, an Audio to MIDI converter with synthesizer system has been implemented. The system with the Fast Fourier Transform (FFT) based power spectrum for peak detection was created in Matlab. The system has been tried and improved to give better results for more and more input music files. The Audio to MIDI converter with synthesizer system has a Graphical User Interface to provide a convenient way to operate the entire system. It provides options to first either select a .wav file or record a .wav file. After a file is either chosen or recorded, its waveform is also displayed and the user is given flexible options to crop the audio as per the needs. Then they can choose whether they want to run the monophonic function or the polyphonic function for note detection as per the kind of file chosen. The user can then decide whether he/she wants to generate a MIDI sequence of the input file, generate a sound from the synthesizer using the MIDI file generated or get sound of a particular instrument with the use of soundfonts to generate the required type of music for the user. All the functionalities are very systematically provided in the GUI, making the system user-friendly and providing them with enough options for customization by providing them the flexibility to enter parameters as and when required.

1.4. Scope

The project includes a software implementation of the proposed system. Initially a .wav file is given as input to the system.

The audio signal is first converted first to the frequency domain on being processed by the software module using the Fast Fourier Transform algorithm. This gives the power spectrum which helps in detection of the frequency and thus gives the note. The frequency found then helps in the generation of MIDI messages.

The project is an implementation of monophonic as well as polyphonic audio signals. But the accuracy decreases as the number of notes being played simultaneously increases and the conversion process goes on becoming complex.

The synthesizer that is implemented performs additive and FM (Frequency Modulation) synthesis to generate the desired synthetic sounds.

Soundfont technology is an implementation of sample-based synthesis. Soundfonts have been used to generate the sound of various acoustic and electric instruments from the generated MIDI sequence, irrespective of the instrument on which that particular piece was originally played.

The proposed system has applications in the following areas:

1. Music industry: To make it easy for composers to get sounds of various instruments or even synthesized sounds to be used.
2. When music is being studied, particularly traditional music or improvised music, where a score does not necessarily exist, transcription can help determine what notes were played.
3. This system could also be used to change musical pieces into a compressed representation such as MIDI files without human intervention, which are widely used in music.

1.5. Organization of Project Report

This report provides a detailed analysis of each and every stage in the development of the project. The report has been divided into five parts and is organized as follows.

Chapter 1: Introduction

This chapter includes the abstract, the introduction and motivation behind this project. It also specifies the problem statement and the scope of the project. Finally, it gives information about the overall organization of the project report.

Chapter 2: Review of Literature

This chapter consists of the explanation on the current technologies and methodologies used in this domain and the techniques and methods used in this project.

Chapter 3: Analysis and Design

This chapter includes the detailed Analysis of the project, which is further broken down into functional and non-functional requirements. It also consists of the proposed system which is implemented in this project. This chapter also focuses on the major design considerations and the design details which have been used in the project. This chapter also includes the hardware and software requirements.

Chapter 4: Implementation and Results

This chapter includes the details related to implementation and explains the results produced after the implementation of the project. This chapter also focuses on the major implementation issues which had been encountered during the implementation and their remedies.

Chapter 5: Testing

This chapter focuses on testing of the Speech Controlled Robotics System. The testing is carried out using different test cases. The results of all the test cases carried out are explained in this chapter.

Chapter 6: Conclusion and Further Work

This chapter majorly focuses on the conclusion and further work which can be carried out in this domain with a view to enhance this project.

2. REVIEW OF LITERATURE

The various concepts used in the implementation of the system are explained below. This involves the various domains and modules involved.

2.1. Music

When discussing music, it is generally accepted that there are two main types of music, Western Tonal and Non-Western music [1]. Here we take Middle Eastern music as one example of Non-Western music. Eastern or Oriental music has its origins in the Middle East and Western Tonal music has its origins in the Western world, Europe and the U.S. The characteristics of these two approaches to music vary greatly, although in recent times more modern Eastern music has been influenced by Western Tonal styles of music [2].

2.1.1. Western Tonal Music

In Western Tonal music there are twelve distinct elements or notes which are denoted by the letters A to G, including sharp (#) and flat (b) notes. These notes are C, C# (also known as Db), D, D# / Eb, E, F, F# / Gb, G, G# / Ab, A, A# / Bb and B.

Each element is a half-step (semitone) higher than the one immediately preceding it. Note that in some cases a given element may be known by two different names – one involving a sharp and the other a flat. These 12 distinct steps compass an octave [3] which corresponds to a doubling of frequency. The frequency range of a given instrument may include several octaves. A note in a specific octave is denoted as C4, A5 where the number denotes the specific octave, and the frequency designation of which may vary under different systems. Note C4 and note C5 are one octave apart, meaning that note C5 is double the frequency or pitch of note C4.

Usually 8 notes, comprising a scale, are selected as the basis for a given piece of music and there are certain rules regarding which 8 notes can be used in a certain scale. For example, a typical Western song in the scale, or played in the key of, C Major will use the tones C, D, E, F, G, A, B and C', where C' is one octave higher than the first C note, as the basis for the song. The first note in a scale is known as the tonic or keynote. These scales or keys are the basic building blocks for a given song.

2.1.2. Eastern Music

Eastern music theory is different from Western Tonal music theory and the music itself can sound very different. Some factors which are significantly different include the use of quarter tones, the absence of chords or harmony, the distinctive rhythms used, and the use of improvisation.

The smallest inter-note relationship in Eastern music is the quarter-tone. If we were to visualize the piano in terms of Eastern music, we would still see the C and the C# piano keys, but we would now see a note halfway in between. These are called quarter tones and are not part of Western Tonal music. Instead of having 12 possible tones to use as the basis of building a scale, Eastern music has twice as many possible tones to choose from [2]. It would be impossible to play an Eastern song that uses quarter tones on many standard Western instruments such as a piano. It is possible to tune some Western instruments to play music with quarter tones, but the standard instrument tuned for normal Western symphonic or concert band music would not be capable of playing quarter tones.

2.2. Domain Explanation

This project is mainly lies under 2 domains viz. Digital Signal Processing (DSP) and Music Information Retrieval (MIR).

2.2.1. Digital Signal Processing

Digital signal processing (DSP) is the mathematical manipulation of an information signal to modify or improve it in some way. It is characterized by the representation of discrete time, discrete frequency, or other discrete domain signals by a sequence of numbers or symbols and the processing of these signals.

The goal of DSP is usually to measure, filter and/or compress continuous real-world analog signals. The first step is usually to convert the signal from an analog to a digital form, by sampling and then digitizing it using an analog-to-digital converter (ADC), which turns the analog signal into a stream of numbers. However, often, the required output signal is another analog output signal, which requires a digital-to-analog converter (DAC). Even if this process is more complex than analog processing and has a discrete value range, the application of computational power to digital signal processing allows for many advantages over analog processing in many applications, such as error detection and correction in transmission as well as data compression.

DSP has different sub-domains like time domain (one-dimensional signals), spatial domain (multidimensional signals), frequency domain, and wavelet domains. In time domain the signals are represented as amplitude against time, while in frequency domain they are represented as magnitude or power against frequency.

This project mainly focuses on ‘Frequency Domain Analysis’ of signal. For analysis in the frequency domain, signals are converted from time or space domain to the frequency domain usually through the Fourier transform. The Fourier transform converts the signal information to a magnitude and phase component of each frequency. Often the Fourier transform is converted to the power spectrum, which is the magnitude of each frequency component squared.

The most common purpose for analysis of signals in the frequency domain is analysis of signal properties. The engineer can study the spectrum to determine which frequencies are present in the input signal and which are missing.

There are some commonly used frequency domain transformations. For example, the cepstrum converts a signal to the frequency domain through Fourier transform, takes the logarithm, then applies another Fourier transform. This emphasizes the harmonic structure of the original spectrum. Frequency domain analysis is also called spectrum- or spectral analysis.

2.2.2. Music Information Retrieval

Music information retrieval (MIR) is the interdisciplinary science of retrieving information from music. MIR is a small but growing field of research with many real-world applications. Those involved in MIR may have a background in musicology, psychology, academic music study, signal processing, machine learning or some combination of these.

MIR has many subdomains, out of which this project lies in melody extraction and retrieval. MIR is being used by businesses and academics to categorize, manipulate and even create music. Applications of MIR are as follows:

Recommender systems:

Several recommender systems for music already exist, but surprisingly few are based upon MIR techniques, instead making use of similarity between users or laborious data compilation. Pandora, for example, uses experts to tag the music with particular qualities such as “female singer” or “strong bass line”. Many other systems find users whose listening history is similar and suggests unheard music to the users from their respective collections. MIR techniques for similarity in music are now beginning to form part of such systems.

Track separation and instrument recognition:

Track separation is about extracting the original tracks as recorded, which could have more than one instrument played per track. Instrument recognition is about identifying the instruments involved and/or separating the music into one track per instrument. Various programs have been developed that can separate music into its component tracks without access to the master copy. In this way e.g. karaoke tracks can be created from normal music tracks, though the process is not yet perfect owing to vocals occupying some of the same frequency space as the other instruments.

Automatic music transcription:

Automatic music transcription is the process of converting an audio recording into symbolic notation, such as a score or a MIDI file. This process involves several subtasks, which include multi-pitch detection, onset detection, duration estimation, instrument identification, and the extraction of rhythmic information. This task becomes more difficult with greater numbers of instruments and a greater polyphony level.

Automatic categorization:

Musical genre categorization is a common task for MIR and is the usual task for the yearly Music Information Retrieval Evaluation eXchange (MIREX). Machine learning techniques such as Support Vector Machines tend to perform well, despite the somewhat subjective nature of the classification. Other potential classifications include identifying the artist, the place of origin or the mood of the piece. Where the output is expected to be a number rather than a class, regression analysis is required.

Music generation:

The automatic generation of music is a goal held by many MIR researchers. Attempts have been made with limited success in terms of human appreciation of the results.

2.3. Wave File

The WAVE file format is a subset of Microsoft's RIFF specification for the storage of multimedia files [4]. A RIFF file starts out with a file header followed by a sequence of data chunks. A WAVE file is often just a RIFF file with a single “WAVE” chunk which consists of two subchunks: a “fmt” chunk specifying the data format and a “data” chunk containing the actual sample data. Figure 2-1 shows the Canonical Wave File Format.

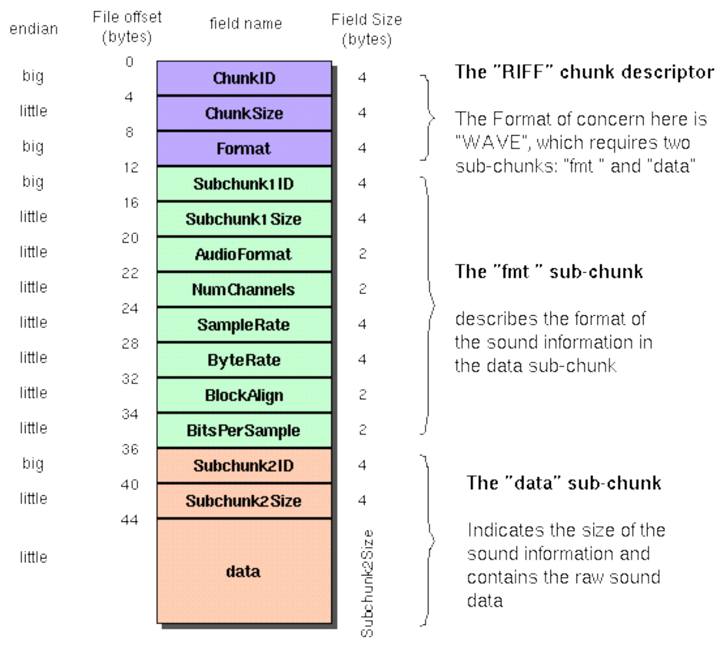


Figure 2-1: Canonical Wave File Format

The canonical WAVE format starts with the RIFF header as shown in Table 2-1:

Table 2-1: Wave File RIFF Header

Offset	Size	Name	Description
0	4	ChunkID	Contains the letters “RIFF” in ASCII form (0x52494646 big-endian form).
4	4	ChunkSize	36 + SubChunk2Size, or more precisely: 4 + (8 + SubChunk1Size) + (8 + SubChunk2Size). This is the size of the rest of the chunk following this number.
8	4	Format	Contains the letters “WAVE” (0x57415645 big-endian form).

The “WAVE” format consists of two subchunks: “fmt” and “data” [5].

The “fmt” subchunk describes the sound data's format as shown in Table 2-2:

Table 2-2: Wave File fmt subchunk

Offset	Size	Name	Description
12	4	Subchunk1ID	Contains the letters “fmt” (0x666d7420 big-endian form).
16	4	Subchunk1Size	16 for PCM. This is the size of the rest of the Subchunk which follows this number.
20	2	AudioFormat	PCM = 1 (i.e. Linear quantization) Values other than 1 indicate some form of compression.
22	2	NumChannels	Mono = 1, Stereo = 2, etc.
24	4	SampleRate	8000, 44100, etc.

28	4	ByteRate	= SampleRate * NumChannels * BitsPerSample/8
32	2	BlockAlign	= NumChannels * BitsPerSample/8. The number of bytes for one sample including all channels.
34	2	BitsPerSample	8 bits = 8, 16 bits = 16, etc.
	2	ExtraParamSize	If PCM, then doesn't exist
	X	ExtraParams	space for extra parameters

The “data” subchunk contains the size of the data and the actual sound. Table 2-3 shows format of data subchunk.

Table 2-3: Wave File data subchunk

Offset	Size	Name	Description
36	4	Subchunk2ID	Contains the letters “data” (0x64617461 big-endian form).
40	4	Subchunk2Size	= NumSamples * NumChannels * BitsPerSample/8. This is the number of bytes in the data.
44	*	Data	The actual sound data.

2.4. MIDI

MIDI (Musical Instrument Digital Interface) is a technical standard that describes a protocol, digital interface and connectors and allows a wide variety of electronic musical instruments, computers and other related devices to connect and communicate with one another. A single MIDI link can carry up to sixteen channels of information, each of which can be routed to a separate device. MIDI carries event messages that specify notation, pitch and velocity, control signals for parameters such as volume, vibrato, audio panning, cues, and clock signals that set and synchronize tempo between multiple devices.

A standard MIDI file is composed of “chunks” [6]. It starts with a header chunk and is followed by one or more track chunks. The header chunk contains data that pertains to the overall file. Each track chunk defines a logical track.

$$SMF\ = < header\_chunk > \ + < track\_chunk > \ [+ < track\_chunk > \ldots]$$

2.4.1. Header Chunk

The header chunk consists of a literal string denoting the header, a length indicator, the format of the MIDI file, the number of tracks in the file, and a timing value specifying delta time units. Numbers larger than one byte are placed most significant byte first.

$$header\_chunk\ =\ \text{“}MThd\text{”} + < header\_length > \ + < format > \ + < n > \ + < division >$$

“MThd” 4 bytes: the literal string MThd, or in hexadecimal notation: 0x4d546864. These four characters at the start of the MIDI file indicate that this is a MIDI file.

<header_length> 4 bytes: length of the header chunk (always 6 bytes long--the size of the next three fields which are considered the header chunk).

<format> 2 bytes:

0 = single track file format

1 = multiple track file format

2 = multiple song file format (i.e., a series of type 0 files)

<n> 2 bytes: number of track chunks that follow the header chunk

<division> 2 bytes: unit of time for delta timing. If the value is positive, then it represents the units per beat. For example, +96 would mean 96 ticks per beat. If the value is negative, delta times are in SMPTE compatible units.

2.4.2. Track Chunk

A track chunk consists of a literal identifier string, a length indicator specifying the size of the track, and actual event data making up the track.

$$track\_chunk\ =\ \text{“}MTrk\text{”} + < length > \ + < track\_event > \ [+ < track\_event > \dots]$$

“MTrk” 4 bytes: the literal string MTrk. This marks the beginning of a track.

<length> 4 bytes: the number of bytes in the track chunk following this number.

<track_event>: a sequenced track event.

Track Event

A track event consists of a delta time since the last event, and one of three types of events.

$$track\_event\ = < v\_time > \ + < midi\_event > \ | \ < meta\_event > \ | \ < sysex\_event >$$

<v_time> : a variable length value specifying the elapsed time (delta time) from the previous event to this event.

<midi_event> : any MIDI channel message such as note-on or note-off. Running status is used in the same manner as it is used between MIDI devices.

<meta_event> : an SMF meta event.

<sysex_event> : an SMF system exclusive event.

Meta Event

Meta events are non-MIDI data of various sorts consisting of a fixed prefix, type indicator, a length field, and actual event data.

$$meta\_event\ =\ 0xFF + < meta\_type > \ + < v\_length > \ + < event\_data\_bytes >$$

<meta_type> 1 byte: meta event types are as shown in Table 2-4:

Table 2-4: MIDI Meta Event Types

Type	**Event**	**Type**	**Event**
0x00	Sequence number	**0x20**	MIDI channel prefix assignment
0x01	Text event	**0x2F**	End of track
0x02	Copyright notice	**0x51**	Tempo setting
0x03	Sequence or track name	**0x54**	SMPTE offset
0x04	Instrument name	**0x58**	Time signature
0x05	Lyric text	**0x59**	Key signature
0x06	Marker text	**0x7F**	Sequencer specific event
0x07	Cue point		

<v_length> : length of meta event data expressed as a variable length value.

<event_data_bytes> : the actual event data.

2.4.3. MIDI Messages

The MIDI specification is divided into two main sections, one is called channel messages and the other side is called system messages [7]. Each MIDI cable sends 16 channels of information and the channel messages are messages that affect each channel independent of one another, the system messages are messages that affect the entire MIDI module. The MIDI messages are as shown in Figure 2-2.

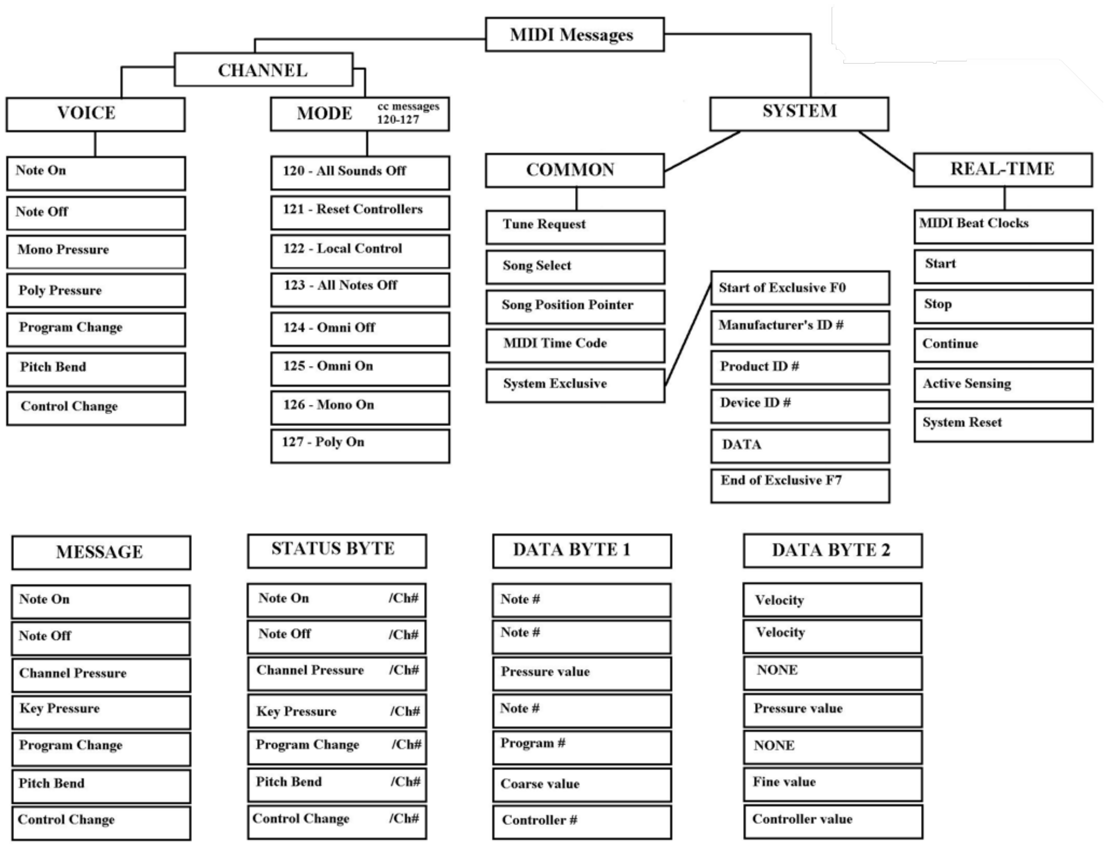


Figure 2-2: Types of MIDI Messages

2.5. SoundFont

SoundFont Technology is a technology that allow you to customize your instrument sounds for MIDI music playback or MIDI sequencing [8]. Developed by Creative Labs and E-MU Systems, SoundFont technology allows the user to improve the sound quality of their MIDI playback by either loading a better sounding General MIDI (GM) wavetable sound-set, or loading new (non-GM) sounds for use in music composition and songwriting. It is used by many sound cards and software synthesizers.

A SoundFont bank (.SF2 or .SBK file) is essentially replacement instruments for the standard GM (General MIDI) instrument set. A General MIDI set (GM) has only 128 (0-127) instruments to choose from. SoundFonts allow custom instruments to play instead of the GM instruments.

2.6. Existing Solution

There have been research and efforts made to perform the audio to MIDI conversion. Some good techniques and algorithms have been realized and in recent years this field of research has gained a lot of importance. These methods and techniques have been studied and understood. The suitable ones have been chosen to start with.

An IEEE transactions paper (2006) by Tao Li and Mitsunori Ogihara does intelligient music information retrieval. This paper introduces Daubechies Wavelet Coefficient Histograms (DWCH) for music feature extraction for music information retrieval [9].

Another approach for the polyphonic note transcription can be formulated as a dynamical Bayesian network [10]. A Generative Model for Music Transcription was presented in that paper which is a special case of switching Kalman filter model.

Then there has been work done in using neural networks for note recognition. Time delay neural networks, self-organizing maps, and linear vector quantization is proposed for the same [11].

A part of the proposed system has been implemented on hardware as well, where the sound wave is received and processed in a digital signal processer with outputting digital sound frequency to a frequency to MIDI (Musical Instrumental Digital Interface) converter in real time [12].

The FFT algorithm has always been there to support a system like this as it enables conversion from the time domain to the frequency domain.

Recognition of piano notes using FRM filters [13] and Automated Classification of Piano–Guitar Notes [14] are also the proposed techniques but these are limited to specific musical instrument(s).

3. ANALYSIS AND DESIGN

The analysis of the system may be done under two distinct heads, i.e., the functional and non-functional requirements. It also consists of the proposed system which is implemented in this project. This chapter also focuses on the major design considerations and the design details which have been used in the project.

3.1. Requirement Analysis

The following project requirements were set at the beginning of the project, identifying the main goals. They are divided into two categories - functional and non-functional requirements, specifying software products required functionality, efficiency and limitations.

3.1.1. Functional Requirements

The following are the Functional Requirements of the system:

1. Take an audio input from user in one of the two formats which are audio file input or recorded audio input. The format should be .wav.
2. Frame size and input audio given to the system.
3. Detect the peaks from FFT and create the matrix that will help to generate the MIDI file.
4. Perform equalization of the notes that have been detected.
5. Generate MIDI file and write it to a .mid file.
6. Display the piano roll after the MIDI sequence is generated.
7. Produce sound from the synthesizer. The synthesizer must include
    a. Additive Synthesis
    b. FM Synthesis
8. The synthesizer module must provide the user with features like amount of type of synthesis, type of waveform (Sine, Square, Sawtooth).

9. Generate sound of any acoustic instrument as per the choice of the user, using soundfonts for the generated MIDI sequence.
10. Provide all the above functionality for monophonic as well as polyphonic music.

3.1.2. Non-Functional Requirements

- **Efficiency:** The system must be efficient in terms of detecting the notes. The noise must be eliminated and amount of false detected notes or undetected notes must be least. Also the sound generated after synthesis or from the soundfonts should be satisfactory to the user.
- **Robustness:** The system must be robust in terms of being able to convert real time inputs like singing or playing of an instrument to MIDI format efficiently and all the conversions should be without any errors.
- **Speed:** The system must convert the input audio file to MIDI format without any delay within allowed time frame even in case of a worst input i.e. real time singing or playing of an instrument.
- **Resource constraints:** The system must perform the task with negligible load on the processor and other system resources.
- **Usability:** The system must be easy to use with a user friendly UI.

3.2. Proposed System

The proposed system takes an audio input from file or recording through microphone. A Fast Fourier Transform (FFT) is applied to the input which converts it from time domain to frequency domain thus availing frequencies of musical notes. The FFT is converted to a power spectrum. The output of the power spectrum has been processed so as to detect notes played in the music. This forms the music transcription part of the system and is the heart of the whole system. The transcribed notes are then used to generate a MIDI message.

Now depending choice of user, output would be provided as follows:

1. The MIDI sequence would be stored in a MIDI file and would be extracted into a location on user's computer.
2. The MIDI sequence would be given to a synthesizer which would play the message with desired synthesized sound.
3. The MIDI sequence would be given to the module that would generate the sound of a desired instrument using soundfonts.

Thus a musician may record the tune with any instrument he or she knows, give it to this software and get the tune in whole lot of different types of instruments or synthesized form.

3.3. Hardware and Software Requirements

This project is a software based implementation and programmed in Matlab. The requirements for the project can be broken down in two categories mainly – the hardware requirements and the software requirements.

3.3.1. Hardware Requirements

- **CPU**: Intel Pentium 4/ Celeron/ Core 2 Duo/ Dual Core/ Core i3/ Core i5/ Core i7 or AMD K series/ Am2900/ Am29000/ Athlon series. (1 GHz or higher).
- **RAM:** 512 MB or higher.
- **Disk space:** 20 MB or more for installation (excluding Matlab software). 1 GB is recommended for storing recordings and soundfonts.
- **External Sound Card** (Optional, for better quality of output and reducing the latency).
- **Microphone** (Optional, for recording the sound of acoustic instruments).

3.3.2. Software Requirements

- **Operating Systems:** Windows XP/ 7/ 8
- Matlab 7.0 or higher with Signal processing toolbox.
- MIDI Toolbox.
- Matlab MIDI - Ken Schutte.
- Intelligent Sound Processing Toolbox.

3.4. Project Design

Figure 3-1 shows the block diagram of the system proposed.

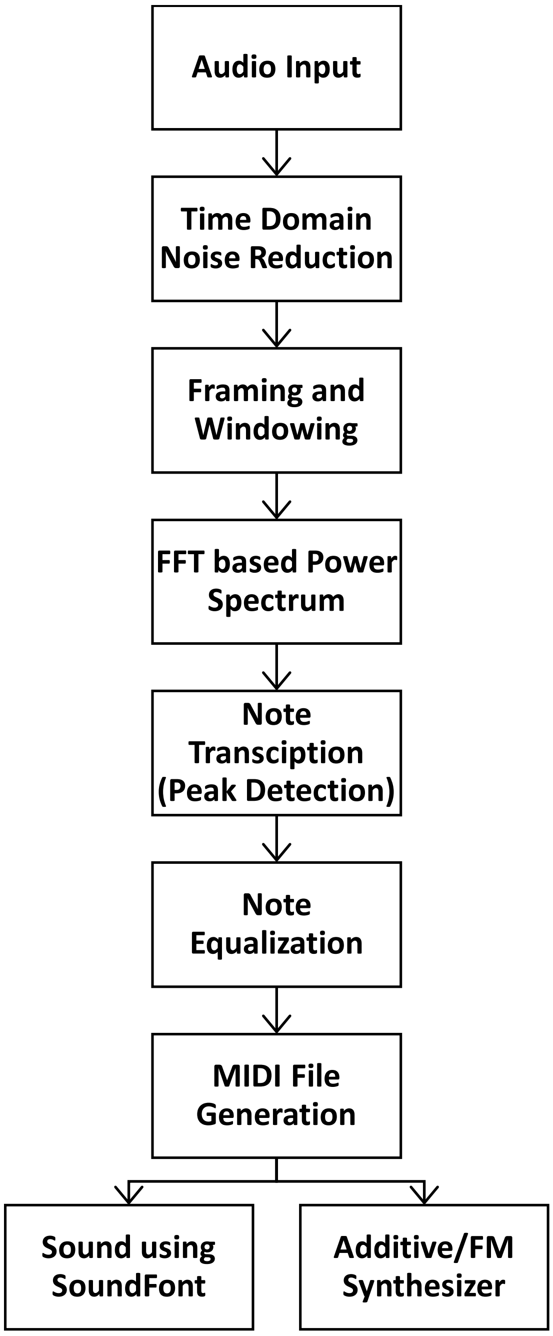


Figure 3-1: Block Diagram of Audio to MIDI Converter with Synthesizer

The diagram shows the various modules involved in the working of the system and the relationship between them. Each of these blocks have been explained below.

3.4.1. Input/Record Audio

There are 2 options available to user. First user can provide the input as an audio file in lossless format (.wav) or user can record the audio using microphone. The sample rate of recorded audio file is 44,100 samples per second with 16 bits per sample.

3.4.2. Time domain noise reduction.

A threshold value is taken from the user. Wherever the signal strength is below this threshold value, its amplitude is reduced to zero. This is to remove the noise that creeps into recordings or music files.

3.4.3. Framing and Windowing

Framing is done based on assumption that in short time period Audio signals do not change much. If the frame is much shorter we don't have enough samples to get a reliable spectral estimate, if it is longer the signal changes too much throughout the frame. In framing the continuous audio signals are divided into N samples and adjacent frames are separated by M samples (M < N). So first frame will start from 0 and will be N samples long. Second frame will start from M samples and will be (N + M) samples long, hence (N – M) samples will overlap. Third frame will start from 2M samples (i.e. M samples after second frame) and (N – M) samples will overlap with fourth frame. This process will be repeated until all audio signals are accommodated on one or more frames. The framing will help in recognizing music notes having small durations.

In windowing, Hanning window function is applied on each frame. This is done to minimize signal discontinuities at start and end of each frame. The concept here is to minimize the spectral distortion by using the window to taper the signal to zero at the beginning and end of each frame [15]. Windowing helps in reducing spectral leakage in the spectrum obtained after applying FFT on each frame. The Hanning window function is:

$$w(n) = 0.5\left(1 - \cos\frac{2\pi n}{N-1}\right);\ 0 \le n \le N-1$$

3.4.4. Fast Fourier Transform (FFT) based power spectrum.

FFT is used to convert time domain signal to frequency domain signal. An FFT computes the DFT and produces exactly the same result as evaluating the DFT definition directly; the only difference is that an FFT is much faster. Let $x_0, \dots, x_{N-1}$ be complex numbers. The DFT is defined by the formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k \frac{n}{N}} \quad ; k = 0, \dots, N-1$$

Evaluating this definition directly requires $O(N^2)$ operations: there are N outputs X_k, and each output requires a sum of N terms. An FFT is method to compute the same results in $O(Nlog(N))$ operations.

The computed FFT is then squared to give the power spectrum. This power spectrum is given to the next module.

3.4.5. Note Transciption (Peak Detection)

Before detecting the peaks a filter bank is applied to the output of the FFT based power spectrum which will block the redundant frequencies and allow only fundamental frequencies of musical notes. There are two parts in peak detection based on monophony and polyphony.

First is monophony, in which only one note played at a time (e.g. Bass instruments). So here peak is detected by finding the frequency at which output of FFT is maximum.

Second is polyphony, in which multiple notes are played simultaneously (e.g. Guitar, Piano etc.). So here multiple peaks are detected by finding the frequencies from the output of the power spectrum based on the amount polyphony.

3.4.6. Equalize notes.

This module optimizes the output given by note transcription and performs performs three main functions to achieve the same. These are described as follows:

1. The user provides a threshold for the number of frames. Now, if a particular note is found to persist for a duration less than or equal to this threshold, it is removed and the previous note is extending for that duration.

2. At times, the harmonics are dominant compared to the original notes played. It checks for such a situation depending on the octave of the previous and the next notes. Thus, an equalization is performed.

3. After notes are shifted or combined with the previous notes, they still continue to be separate in the MIDI sequence. So a proper merger of these messages is performed in this module.

3.4.7. MIDI File Generation

From the frequencies detected from the peak detection module. The MIDI note number is generated. The mapping of frequency to MIDI note is given by:

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

In above equation p is MIDI note number and f is frequency in Hertz. Then the MIDI notes are encoded in a MIDI file.

Velocity for each note is calculated from the time domain signal by taking the maximum amplitude in a frame.

The note start time and end time are calculated by checking the number of frames that each note lasts for and then adding the time for each frame.

3.4.8. Sound using soundfonts (Audio Output)

This module takes the MIDI file generated and gives the user a list of instruments to select from. The user chooses an instrument on which he/she would want the MIDI file to be played. This module then generates the desired instrument’s sound using it’s respective soundfont.

3.4.9. Synthesizer (Audio Output)

The Synthesizer produces a desired sound from the MIDI file generated in above module. The Synthesizer implemented in this project has two types of synthesis viz. Additive Synthesis and FM (frequency modulation synthesis) Synthesis.

Additive synthesis builds sounds by adding together waveforms (which are usually harmonically related). While, FM synthesis is a process that usually involves the use of at least two signal generators (sine-wave oscillators, commonly referred to as “operators” in FM-only synthesizers) to create and modify a voice. The user is provided with an arsenal of options to use the waveforms, thresholds, values of constants that will be used to synthesize the desired sound. Enough flexibility is provided to synthesize any type of sound for the user.

ADSR Envelope

ADSR stands for Attack Decay Sustain Release. ADSR envelope gives the user an option to manipulate the sonic characteristics of the output of the synthesizer. The contour of ADSR envelope is specified using four parameters as shown in Figure 3-2:

- **Attack time:** is the time taken for initial run-up of level from nil to peak, beginning when the key is first pressed.
- **Decay time:** is the time taken for the subsequent run down from the attack level to the designated sustain level.
- **Sustain level:** is the level during the main sequence of the sound's duration, until the key is released.
- **Release time:** is the time taken for the level to decay from the sustain level to zero after the key is released.

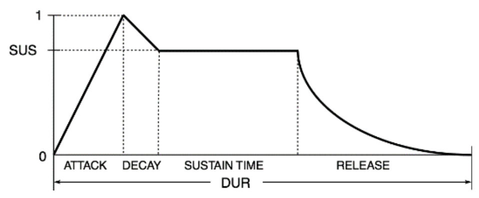


Figure 3-2: ADSR Envelope

4. IMPLEMENTATION AND RESULTS

This chapter includes the details related to implementation and explains the results produced after the implementation of the project. This chapter also highlights the major implementation decisions that were taken and what lead to them.

4.1. Implementation Details

The implementation of the system depends on the design completely. Like seen in the previous section the development advances through the different phases in the software development. Each of the phase is implemented using various techniques and algorithms. The entire implementation is done in Matlab. The implementation of each module proceeds as follows:

Step 1: Take the input.

The user can select a file by browsing through the folders. This would be a pre-recorded music piece lying on the user’s computer and it should be in the Wave format (.wav).

The basic code snippet to open and read this Wave file is:

```
[y,Fs] = wavread('57.wav');
```

This reads the Wave file named ’57’ into the variable y and the sampling frequency gets stored in Fs.

The other option that has been provided is that of recording. The recording option allows users to record a musical piece of their choice using a microphone directly into the system. The recording is done at a sampling frequency of 44100 Hz and at 16 bits/sample.

The code snippet which allows us to perform the recording is given below. Here, recObj creates an object of the recorder. The record() function starts the recording and the stop() function stops the recording. Finally getaudiodata() function retrieves the recorded data and stores it into any variable.

```
recObj= audiorecorder(fs, bits, 1);
record(recObj);
```

```
stop(recObj);
myRecording = getaudiodata(recObj);
```

Now, in both the cases the waveform is displayed in GUI and the user is given the choice to crop the audio by selecting appropriate points on the waveform. An option to undo the most recent crop is also provided.

Step 2: Time domain noise reduction.

Here, a threshold value is taken from the user. Now, wherever the input signal has an amplitude less than this threshold value, the amplitude at those points is made zero. We are thus performing hard clipping.

The function which performs this is:

```
function left=clip(left,thr)
    for i=1:length(left)
          if abs(left(i))<thr
                left(i)=0;
          end
    end
end
```

Step 3: Framing and Windowing

An appropriate size frame is selected. This serves as a measure as to the number of samples that should be considered every time and forms the basic unit for evaluation of the frequency at that point. All the processing is performed on this frame to obtain the pitch, velocity.

A hanning window is used for windowing. Each frame is multiplied with this window before the FFT and the power spectrum are calculated. Windowing reduces spectral leakage and avoids the peaks that would otherwise have been present in the FFT due to noise.

```
w=hanning(length(y));
```

The above line of code stores a hanning window of size equal to the length of signal y in the variable w.

Step 4: Music transcription (Detect monophonic/polyphonic notes)

First we pad the input audio file that is generated after the previous step. This padding with zeros is done so that as we move the frame on the signal, in the last iteration the sample must not be less than the frame size. That is why this adjustment needs to be made. Figure 4-1 shows the block diagram of Note Transcription module.

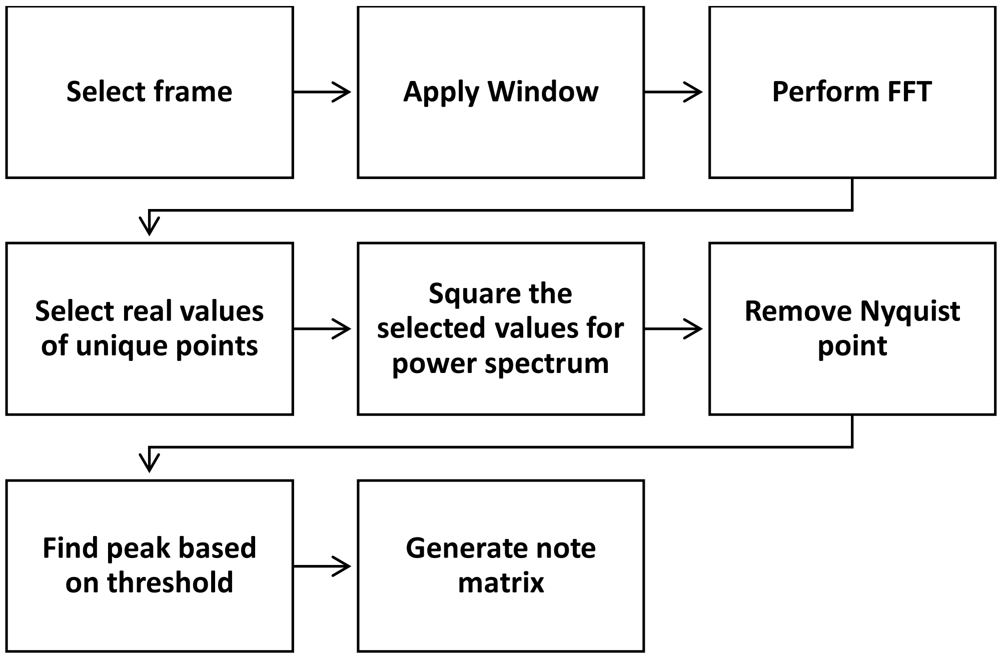

Select frame
Apply Window
Perform FFT
Select real values of unique points
Square the selected values for power spectrum
Remove Nyquist point
Find peak based on threshold
Generate note matrix

Figure 4-1: Block Diagram of Note Transcription

Monophonic detection:

For detecting the notes in the selected monophonic piece of music, we work on the frames individually. First we multiply the frame of the audio with the frame of the window that is explained above. The snippet for the same is given below.

```
wframe = frame.*w;
```

Here wframe stores the windowed signal.

After applying window we perform Fast Fourier Transform (FFT) on the windowed signal to convert it from time domain to frequency domain. The snippet for the same is given below.

```
p = fft(wframe);
```

Now p stores the frequency domain signal. As a characteristic of FFT, it produces a mirrored signal which means that the signal is replicated along a center line on both the sides. Due to this we only use left part of the signal. nUniquePts is one-half the length of the original signal.

```
p = p(1:nUniquePts);
```

We then compute the power spectrum and the following code does that:

```
p = abs(p);
p = p/fl;
p = p.^2;
if rem(fl, 2)
p(2:end) = p(2:end)*2;
else
p(2:end-1) = p(2:end-1)*2;
end
[max_value, index] = max(p(:));
maxvalarr(i) = max(frame);
notearray(i) = freqArray(index);
i = i+1;
s1 = s1+n-1;
```

The if…else condition here removes the Nyquist point. Here, notearray[] stores the pitch or the frequency for each frame I, and maxvalarr[] is a vector that stores the maximum amplitude of the frame i. All of the above processing is done on each frame and so we run the above code inside a while loop and have the terminating condition i<=k. The variable k is the count of, the number of times the window has to be shifted.

The function detectMono() written to detect notes in a monophonic input file generates or returns a matrix M which later gets used to generate the output MIDI file.

The basic format for this matrix is as shown below:

```
M - a matrix containing a list of notes, ordered by start time.
%     column values are:
%      1     2     3   4    5   6
%     [track chan nn  vel   t1  t2
%   endtime - time of end of track message
```

T1 gives the start time and t2 gives the end time for that particular MIDI message.

The code that appropriately creates this matrix M from the processing that we did above is as follows:

```
st = 0; %Start Time
x=1;
i=1;
while i < k
    M(x,3)=round(12*log2(notearray(i)/440)+69);
    M(x,4)=(1+mapminmax(maxvalarr(i)))*60;
    nt=0; %note time
    while i<k && (notearray(i)==notearray(i+1))
        i=i+1;
        nt = nt+ftime;
    end
    nt = nt+ftime;

    M(x,5)=st;
    M(x,6)=st+nt;
    st=st+nt;
    x=x+1;
    i=i+1;
end
M(:,1) = 1;          % all in track 1
M(:,2) = 1;          % all in channel 1
```

As we see from the code above, in column 3 of the matrix we store the note number, which is calculated using the formula shown above. Column 4 has the amplitude and thus the velocity information. The variable nt specifies the note time for each of the note being detected. Variable st specifies the start time of each note. Thus, using st and nt we have also calculated the end time and stored that value in column 6 of matrix M. Each row in this matrix now represents the details for each note being played in the input audio file.

Polyphonic detection

The algorithm for polyphonic detection exactly follows the sequence of events in monophonic detection. Here, we take the amount of polyphony from the user.

All the above steps are iterated for times equal to the amount of polyphony. What this signifies is that if for example the amount of polyphony specified by the user is 2, then from every frame we will take the highest and the second highest amplitudes and put the frequencies at these two points in the notearray[].

Step 5: Equalize notes

We have a function equalizeNotes() which performs the following three operations:

1. The user provides a threshold for the number of frames. Now, if a particular note is found to persist for a duration less than or equal to this threshold, it is removed and the previous note is extending for that duration.

2. At times, the harmonics are dominant compared to the original notes played. It checks for such a situation depending on the octave of the previous and the next notes. Thus, an equalization is performed.

3. After notes are shifted or combined with the previous notes, they still continue to be separate in the MIDI sequence. So a proper merger of these messages is performed in this module.

The function equalizeNotes() is as follows:

```
function M=equalizeNotes(M,n,Fs,x)
i=1;
while(i<length(M(:,1)))
    if M(i,6)-M(i,5)<=x*(n/Fs)
        M(i+1,5)=M(i,5);
        M(i,:)=[];
    end
    i=i+1;
end
for i=2:length(M(:,3))
    if (M(i-1,3)-M(i,3))>10
        M(i,3)=M(i,3)+12;
    end
    if (M(i-1,3)-M(i,3))<-10
        M(i,3)=M(i,3)-12;
    end
end
M=sortrows(M,5);
i=1;
while(i<length(M(:,3)))
    if M(i,3)==M(i+1,3)
        M(i,6)=M(i+1,6);
        M(i+1,:)=[];
    end
    i=i+1;
end
end
```

Variable x is the threshold given by the user.

Step 6: Generate MIDI File.

The output of step 5, i.e. matrix M is given to matrix2midi function, which is a part of matlab-midi-master toolbox. matrix2midi method returns a matrix which is in format to be given to the next method, which is writemidi. Matrix2midi method generates a midi matlab structure from a matrix specifying a list of notes. The structure output can then be used by writemidi.m.

The output of matrix2midi method is given to writemidi method, which is also a method of matlab-midi-master toolbox. writemidi method writes the input matrix into a midi file with a proper format as explained above.

Step 7: Synthesizer

There are two types of synthesizer option provided for the user to synthesize his/her generated midi file. The two synthesizers are Additive synthesizer and Frequency Modulation (FM) synthesizer.

Additive synthesizer

The user is given the option of three types of waves, which are Sin wave, Square wave and Sawtooth wave. In additive synthesis the output is produce by adding all the waves. The user can also specify the parameters for Attack-Decay-Sustain-Release (ADSR) envelope. These parameters are used to generate the sounds note by note which are appended into an array in a loop. This array, which is generated contains the synthesized audio.

The function for additive synthesis is as follows:

```
function
monoSynthAdd(M,Fs,b,c,wavef1,wavef2,wavef3,At,Av,Dt,Dv,St,Rt)
    y=0;
    x1=str2func(wavef1);
    x2=str2func(wavef2);
    x3=str2func(wavef3);
    for i=1:length(M(:,1))
        t=M(i,5):1/Fs:M(i,6);
        f= (2^((M(i,3)-69)/12))*(440);
        Ae=linspace(0,Av,At*length(t));
        De=linspace(Av,Dv,Dt*length(t));
        Se=linspace(Dv,Dv,St*length(t));
        Re=linspace(Dv,0,Rt*length(t));
        ADSR=[Ae De Se Re];
        ADSR=padarray(ADSR,[0(length(t)-length(ADSR))],0,     …
post');
        a=M(i,4)/127;
        b=a*b;
        c=a*c;
       y = [y (a*x1(f*2*pi*t)+b*x2(f*pi*t)+c*x3(f*4*pi*t)) .* …
     ADSR(1,:)];
    end
    figure(10);
    plot(ADSR);
    soundsc(y,Fs);
end
```

A special case of Additive synthesizer is also implemented which produces sound similar to a woodwind instrument called Clarinet. The code is as follows:

```
w=2*pi*f*t;
        y = [y (a*(sin(w)+ 0.75*sin(3*w)+ 0.5*sin(5*w)+ …
0.14*sin(7*w)+0.5*sin(9*w)+0.14*sin(11*w)+0.17*sin(13*w))).*AD
SR(1,:)];
```

Frequency Modulation Synthesizer

In audio and music, Frequency Modulation Synthesis (or FM synthesis) is a form of audio synthesis where a simple waveform (such as a square, triangle, or sawtooth) is changed by modulating its frequency with a modulator frequency that is also in the audio range, resulting in a more complex waveform and a different-sounding tone. Here too the user is given an option of three waves, which are sin, square and sawtooth. In FM synthesis the output of one function is modulated with respect to another function. These function are generally the waveforms described above. The function for FM synthesis is as follows:

```
function monoSynthFM(M,Fs,b,fc,wavef1,wavef2,At,Av,Dt,Dv,St,Rt)
    y=0;
    x1=str2func(wavef1);
    x2=str2func(wavef2);
    for i=1:length(M(:,1))
        t=M(i,5):1/Fs:M(i,6);
        f= (2^((M(i,3)-69)/12))*(440);
        Ae=linspace(0,Av,At*length(t));
        De=linspace(Av,Dv,Dt*length(t));
        Se=linspace(Dv,Dv,St*length(t));
        Re=linspace(Dv,0,Rt*length(t));
        ADSR=[Ae De Se Re];
        ADSR=padarray(ADSR,[0     (length(t)-length(ADSR))],0,…
'post');
        a=M(i,4)/127;
        b=a*b;
        I_env = 5.*ADSR(1,:);
      y = [y (b*x2(2*fc*pi*t+I_env(1,:).*(a*x1(f*2*pi*t)))).* …
ADSR(1,:)];
    end
    figure(10);
    plot(ADSR);
    soundsc(y,Fs);
end
```

These two synthesizers have also been implemented for polyphonic input files. The code is almost similar to those for monophonic, but there is an extra loop to add the sounds due to multiple notes being played at the same time. The sound due to these notes is added into the output vector y at every instance and then the loop keeps appending to y. Thus, the final synthesized sound for polyphonic music is obtained in y.

Step 8: Display piano roll.

Piano roll shows a plot of the MIDI notes versus the time axis. Thus, looking at it a musician gets an idea of what kind of music is being played.

First function that we have used to display the piano roll is piano_roll() from the matlab-midi-master toolbox. To use this function, we convert the matrix M that we earlier created to an appropriate format using the function matrix2midi(). The output is stored in midi_new. This is then given as an input to the function to generate Notes. Notes is then given as an input to the function piano_roll() to generate the desired piano roll.

In the output given by this function, the velocity information is conveyed by different colors.

The call to this function and the plotting of the piano roll is as follows:

```
%% compute piano-roll:
[PR,t,nn] = piano_roll(Notes,1);
```

```
%% display piano-roll:
figure;
imagesc(t,nn,PR);
axis xy;
xlabel('time (sec)');
ylabel('note number');
```

The other function that has been used to display the piano roll is the pianoroll() function from the midi toolbox.

Here, midi2nmat() function takes the earlier generated MIDI file as the input and stores the output in a variable nmat. This is then given as input to the function pianoroll(). Also it should be noted here, that two separate plots for the notes and velocities are created. The call to this function is as below. The function also plots the piano roll.

```
nmat = midi2nmat('testout.mid');
pianoroll(nmat,'Test','sec','vel');
```

Step 9: Generate sound from sounfonts

Here, we use the generated MIDI file to produce sounds of some desired instruments using soundfonts. For this we need to put in various soundfonts, which we have on our system. The isp_midisynth function from the ISP toolbox is used to perform this function. The code that implements this functionality is as given below:

```
ispmidi = isp_midiread('testout.mid');
ispmidi.instruments=zeros(1,4);
ispmidi.controller=zeros(1,5);
[wav,      fs]=isp_midisynth(ispmidi,      'soundfont',        …
'stavi_violin.sf2');
soundsc(wav,fs);
```

isp_midiread() first reads the midi file that we generated earlier. This then goes as an input to the isp_midisynth() function along with the type of soundfont that one desires to use. The function returns the output in wav and the sampling frequency for this audio in fs. The soundsc function is then used to play the sound thus generated using the specified soundfont.

5. TESTING

This chapter explains the tests that the system was put through. The modules were initially tested individually. On attaining successful results, modules were combined to test them as a group. The test results for the modules are as follows.

The home screen contains options to input file or record the audio file. After selecting the input file from one of the two methods the user can operate on the input depending on the type of audio file that is monophonic or polyphonic. Figure 5-1 shows the home screen of the application.

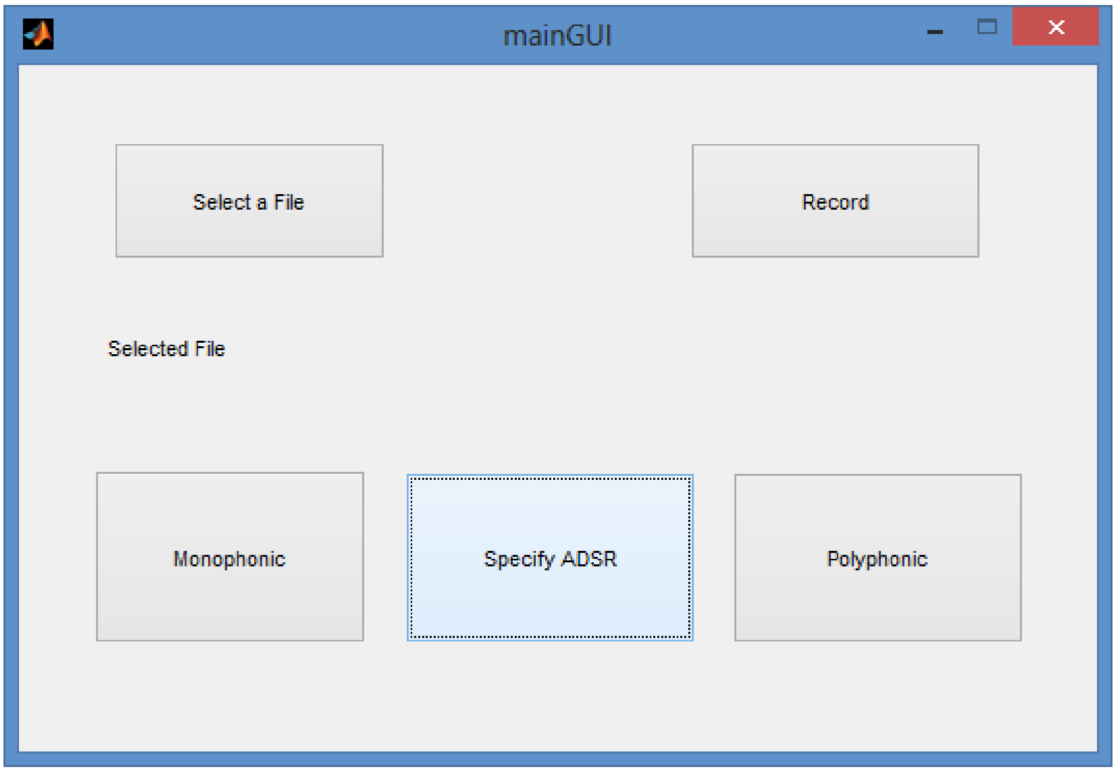


Figure 5-1: Main Screen of the Application

Figure 5-2 shows the screen for browsing the file for input. Figure 5-3 shows the screen with a waveform of the input file specified.

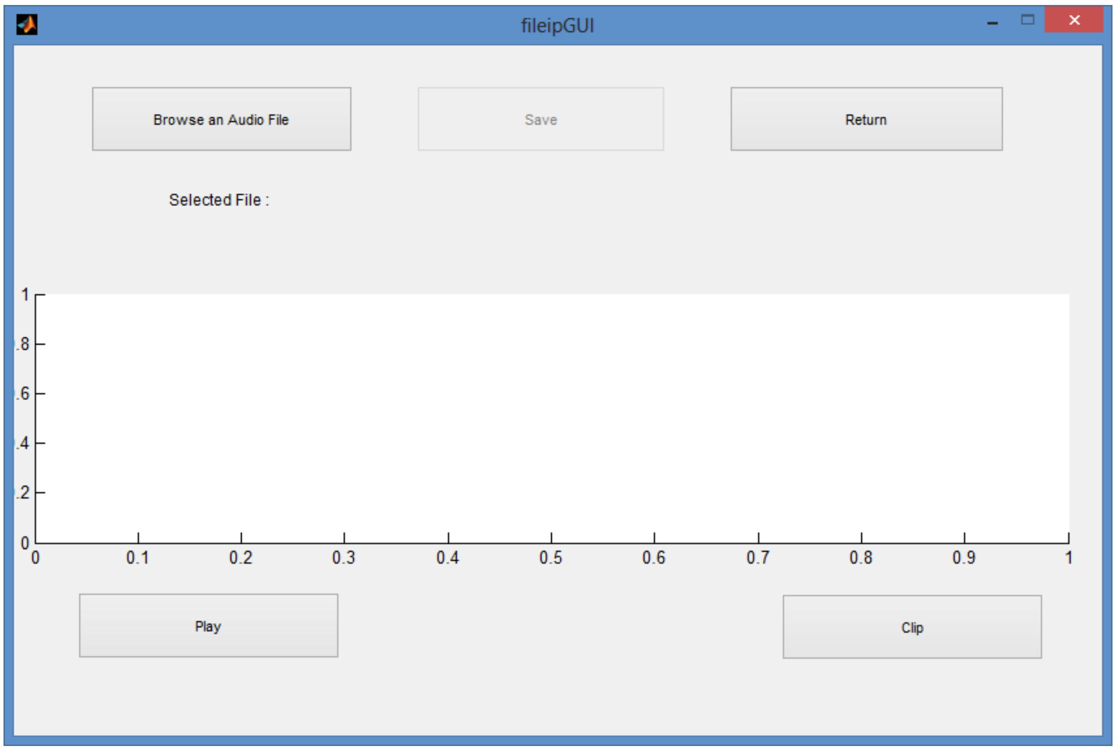
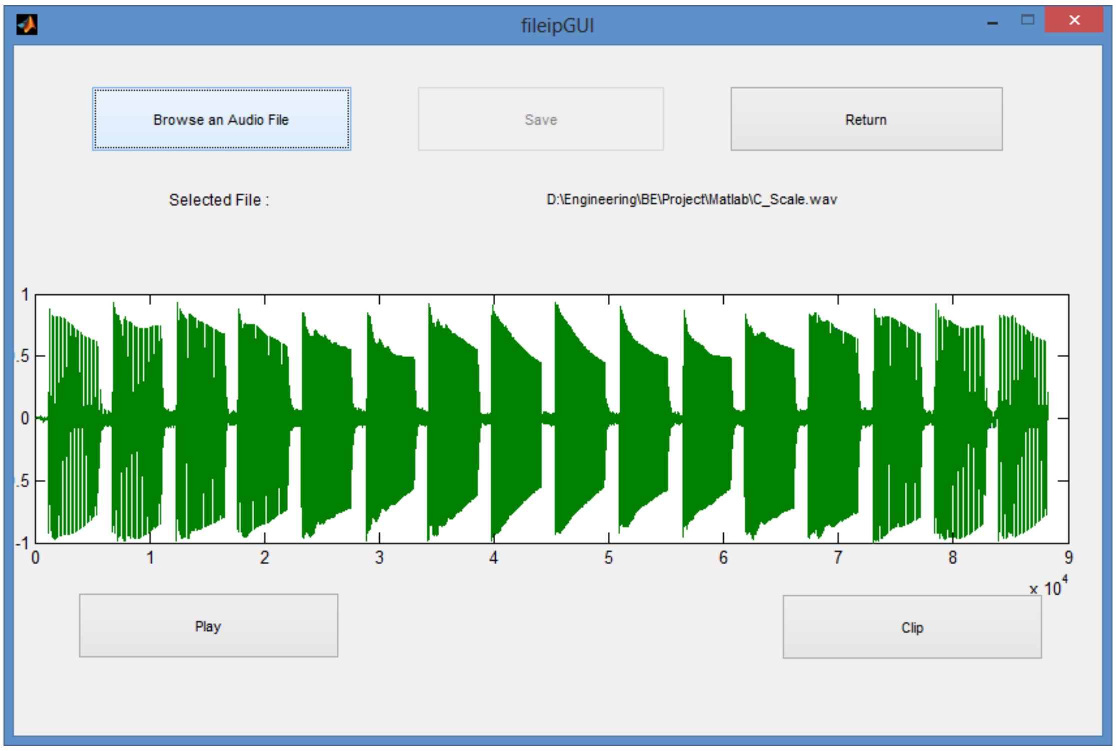


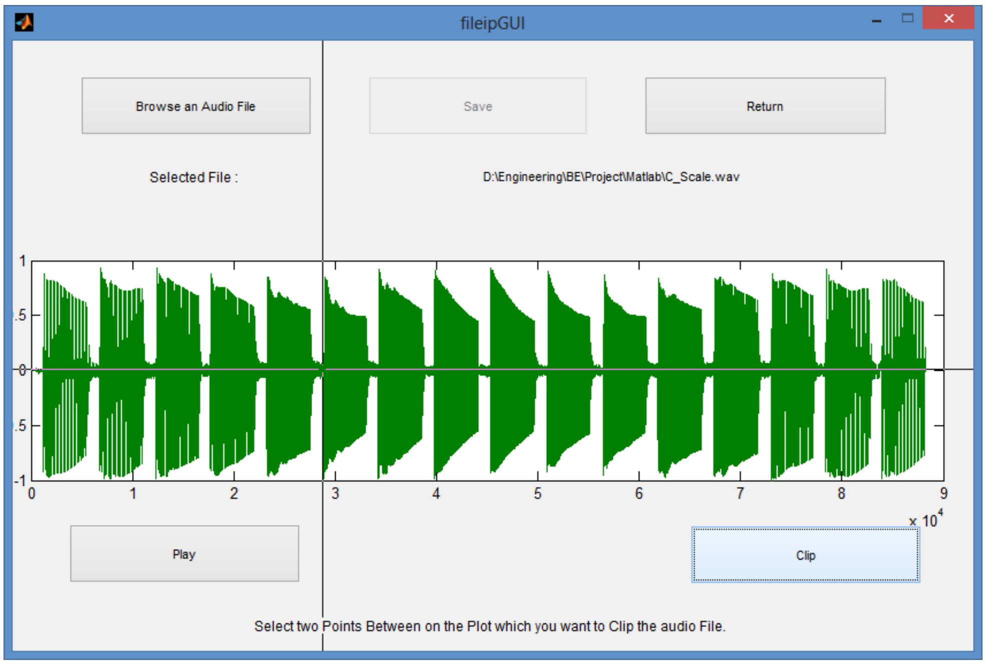
Figure 5-2: File Input Screen



Selected File : D:\Engineering\BE\Project\Matlab\C_Scale.wav

Figure 5-3: Audio file opened in File Input Screen

The user is given the ability to crop the audio by selecting two points on the plot between which the audio is cropped. When clip button is clicked a crosshair is displayed as shown in Figure 5-4.



Select two Points Between on the Plot which you want to Clip the audio File.

Figure 5-4: Clip Function in File Input Screen

The screen for recording the audio file for input to the system is shown in Figure 5-5:

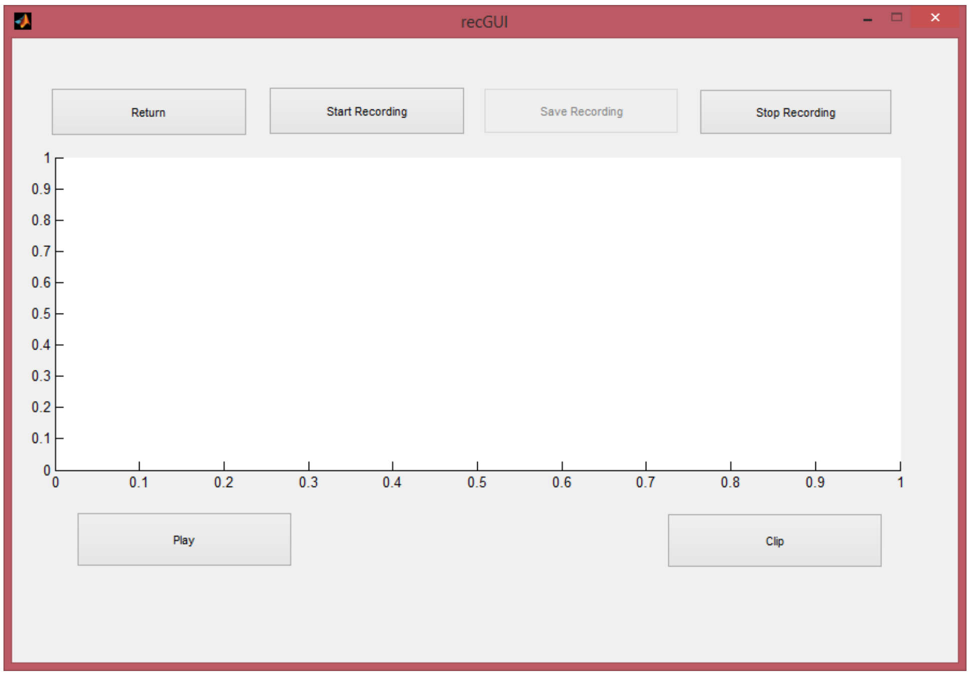


Figure 5-5: Audio Recording Screen

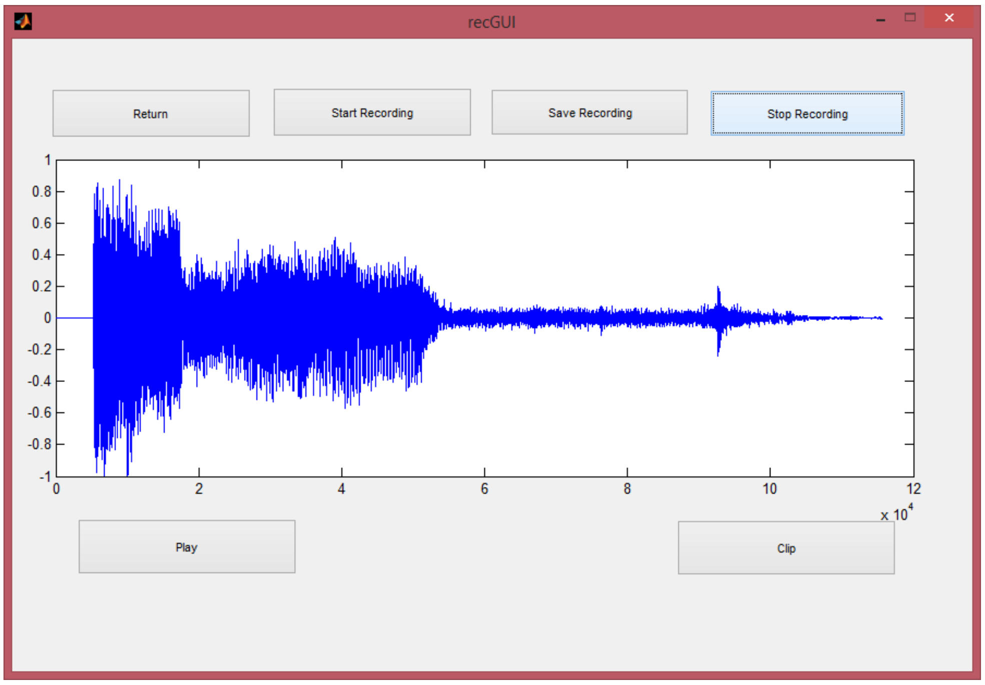


Figure 5-6: Recorded Audio file in Audio Recoded Screen

The user is provided the ability to crop the audio as well on this screen as shown in Figure 5-6. After selecting and cropping audio file user needs to select the ADSR envelope. Figure 5-7 shows the ADSR Screen.

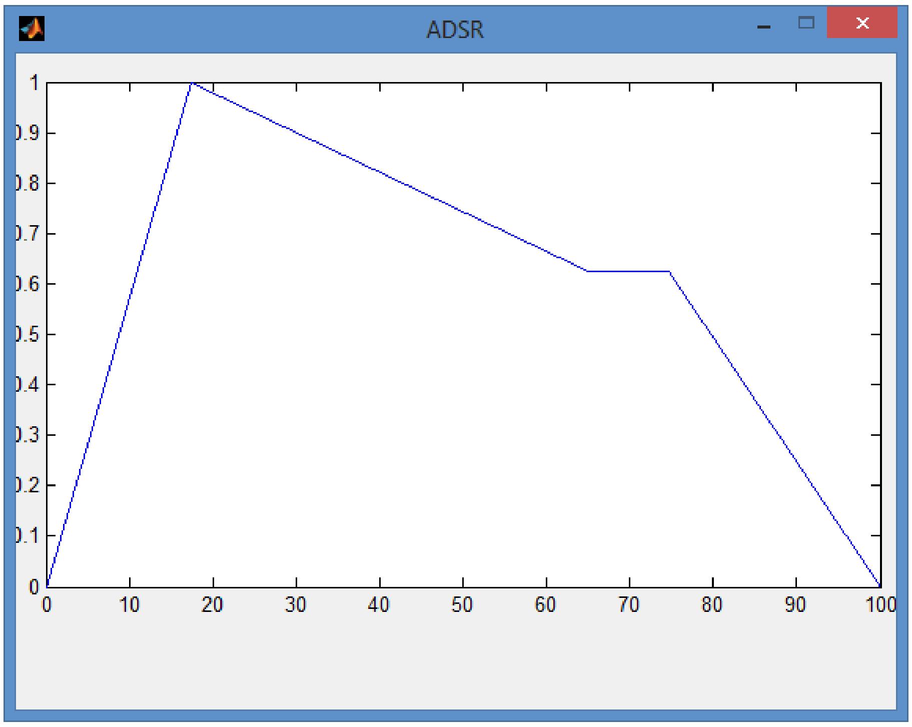
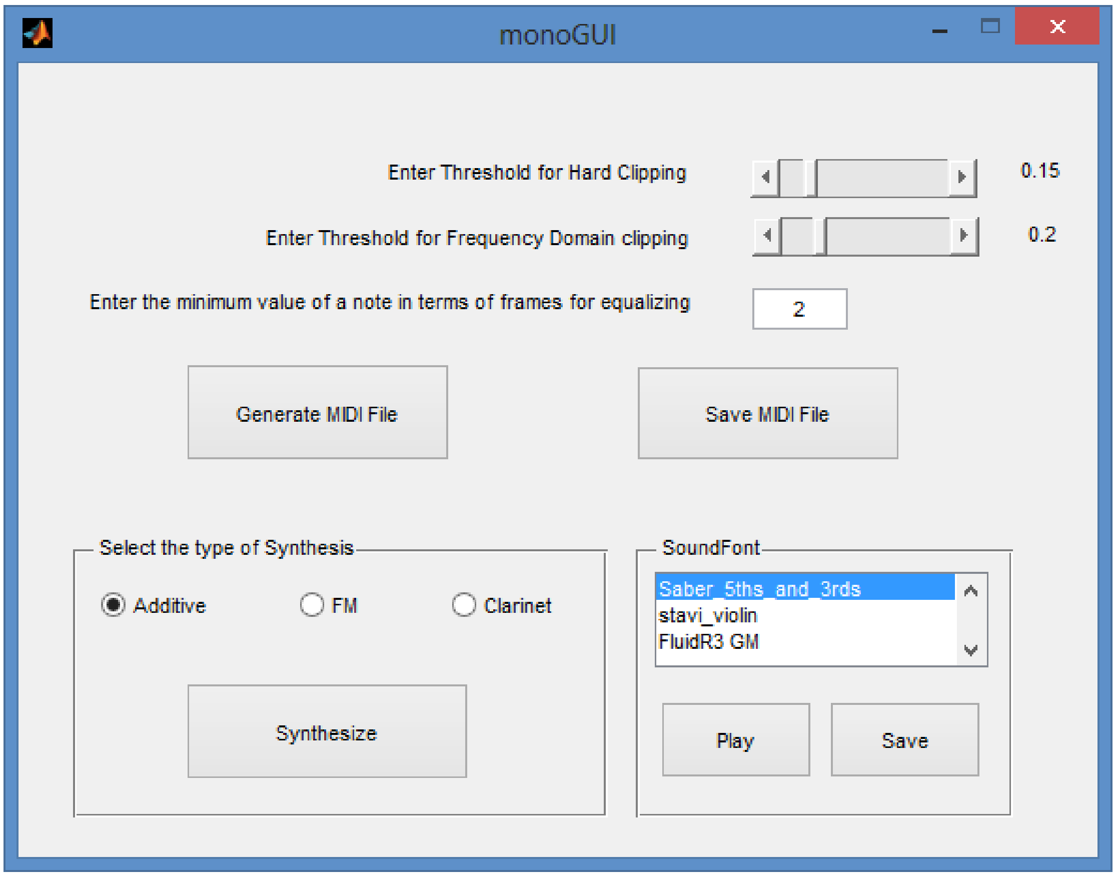


Figure 5-7: ADSR Envelope Input Screen

The screen for generating and synthesizing the monophonic MIDI file is as given follows. In this screen the user is given the ability to provide the threshold for hard clipping and threshold for clipping in frequency domain. After generating the midi file the user can synthesize the same using one of the three methods provided, which are additive, FM, and Clarinet. The generated MIDI file can also be synthesized using sound fonts, which gives the output as sound of various instruments listed. Figure 5-8 shows the monophonic note detection screen.


monoGUI

Enter Threshold for Hard Clipping 0.15

Enter Threshold for Frequency Domain clipping 0.2

Enter the minimum value of a note in terms of frames for equalizing 2

Select the type of Synthesis: [✓ Additive] FM Clarinet

SoundFont: [✓ Saber_5ths_and_3rds] stavi_violin FluidR3 GM

Figure 5-8: Monophonic Note Detection Screen

The piano roll after generating MIDI file is as shown in Figure 5-9.

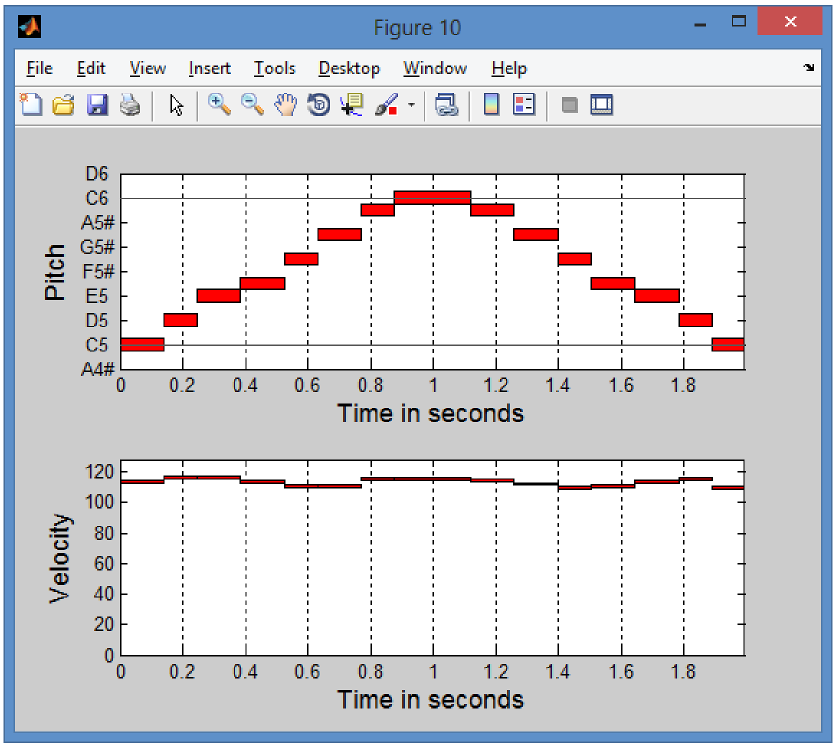
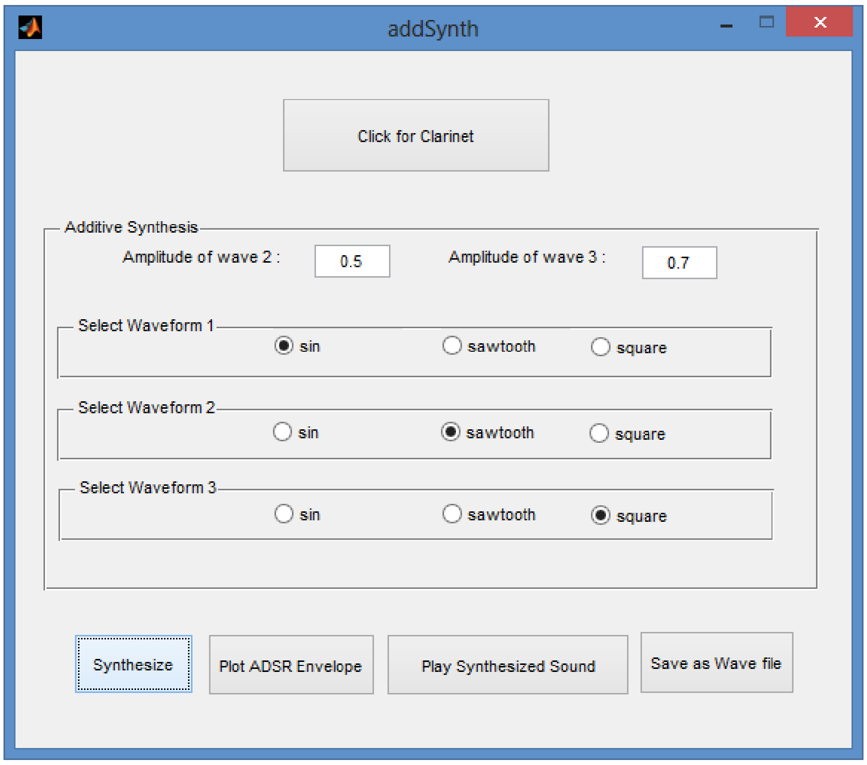


Figure 5-9: Monophonic Piano-roll

In additive synthesis the user is given the ability to provide inputs for amplitude of waves and types of waves. There is a toggle button for generation of sound that resembles the sound of a woodwind instrument called Clarinet. Clarinet sound is generated using additive synthesis but here user cannot select the type of waves or amplitude of wave. Figure 5-10 shows the screen for additive synthesizer where user can vary the various parameters to modulate the output sound.



Additive Synthesis

Amplitude of wave 2 : 0.5 Amplitude of wave 3 : 0.7

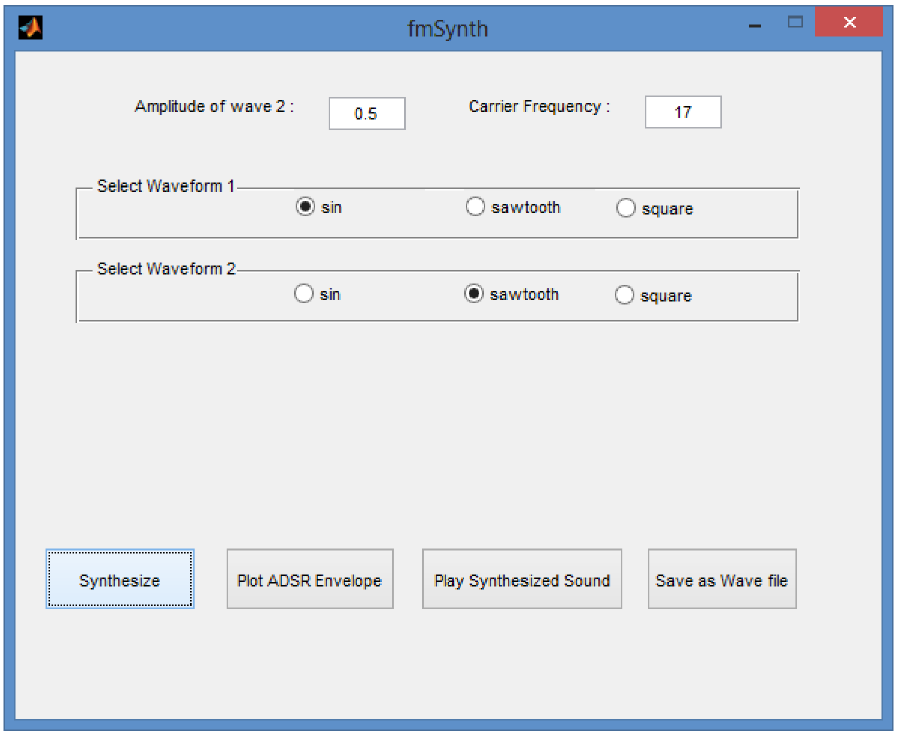
Select Waveform 1: [✓ sin] sawtooth square

Select Waveform 2: sin [✓ sawtooth] square

Select Waveform 3: sin sawtooth [✓ square]

Figure 5-10: Additive Synthesizer

The screen for FM synthesis is as shown in Figure 5-11.



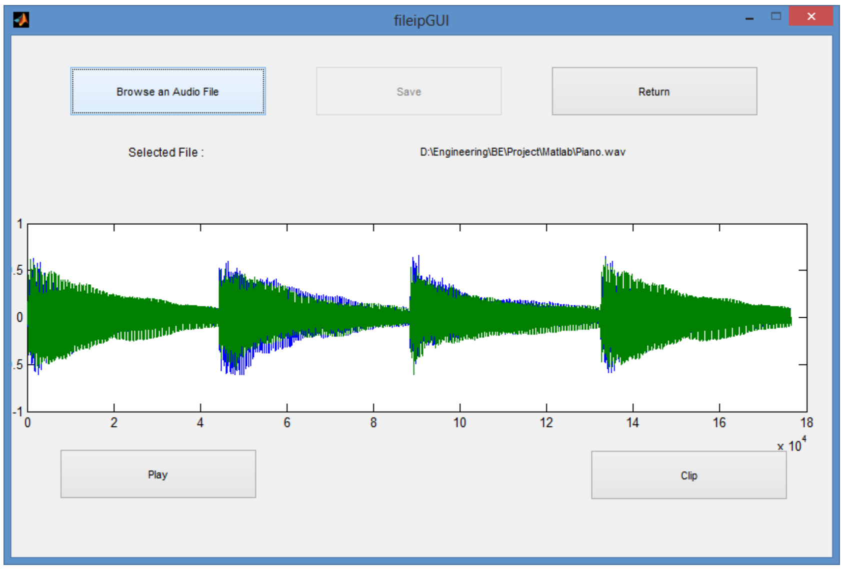
Amplitude of wave 2 : 0.5 Carrier Frequency : 17

Select Waveform 1: [✓ sin] sawtooth square

Select Waveform 2: sin [✓ sawtooth] square

Figure 5-11: FM Synthesizer

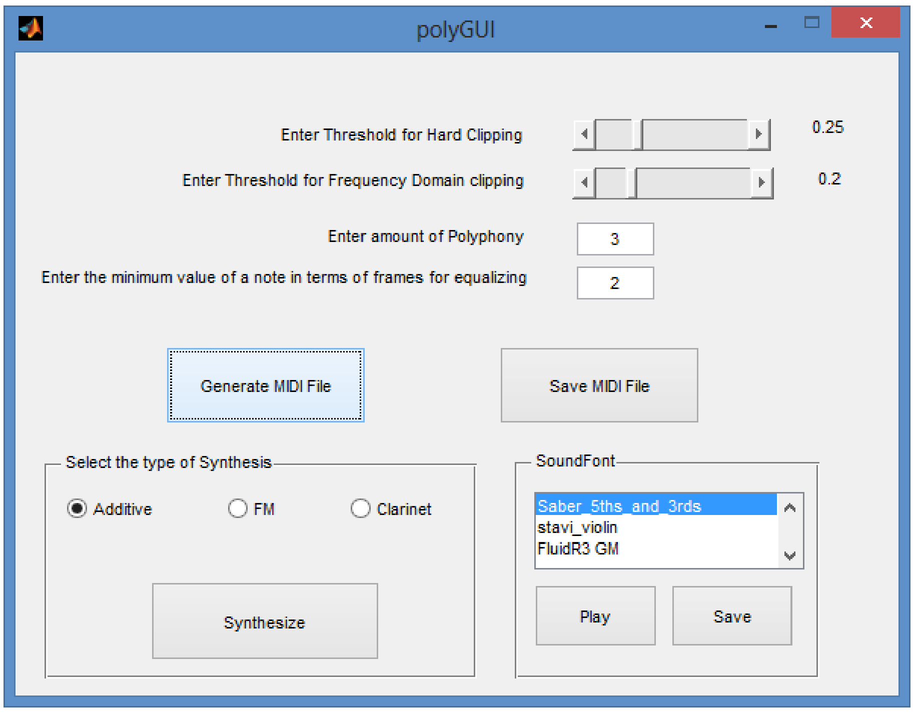
Figure 5-12 is the screen for polyphonic type of input:



Selected File : D:\Engineering\BE\Project\Matlab\Piano.wav

Figure 5-12: Polyphonic File Input Screen

In polyphonic note transcription the user is provided with one more option i.e. amount of polyphony. This will decide how many notes should be recognized in each of the frames. Figure 5-13 shows the polyphonic note detection screen and Figure 5-14 shows the piano roll of the MIDI file generated from polyphonic note detection.



Enter Threshold for Hard Clipping 0.25

Enter Threshold for Frequency Domain clipping 0.2

Enter amount of Polyphony 3

Enter the minimum value of a note in terms of frames for equalizing 2

Select the type of Synthesis: [✓ Additive] FM Clarinet

SoundFont: [✓ Saber_5ths_and_3rds] stavi_violin FluidR3 GM

Figure 5-13: Polyphonic Note Detection Screen

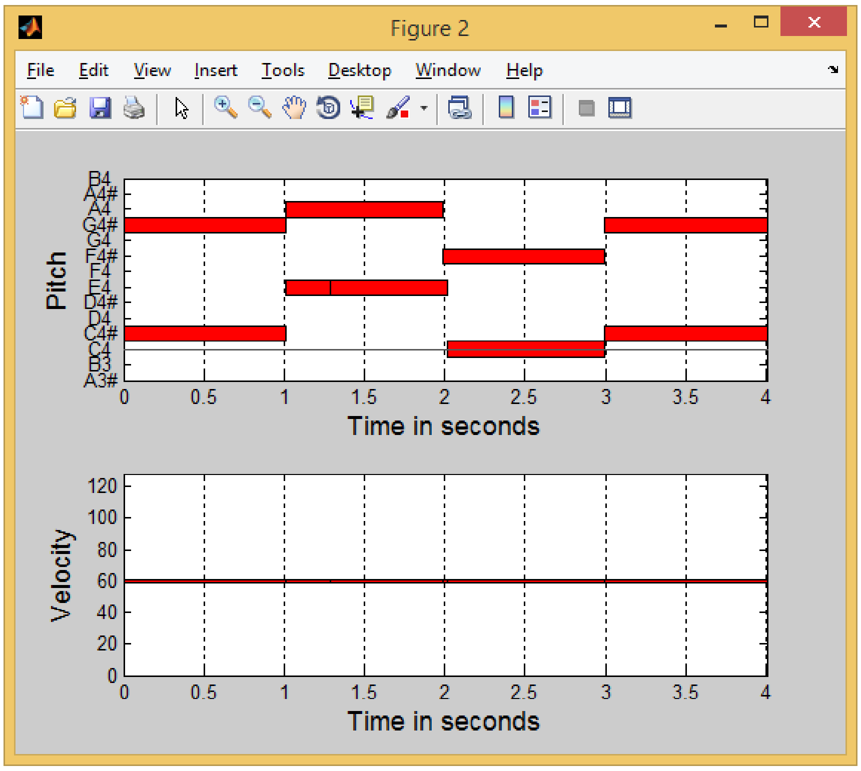


Figure 5-14: Polyphonic Piano-roll

Figure 5-15, Figure 5-16 and Figure 5-17 shows the piano rolls of the test cases:

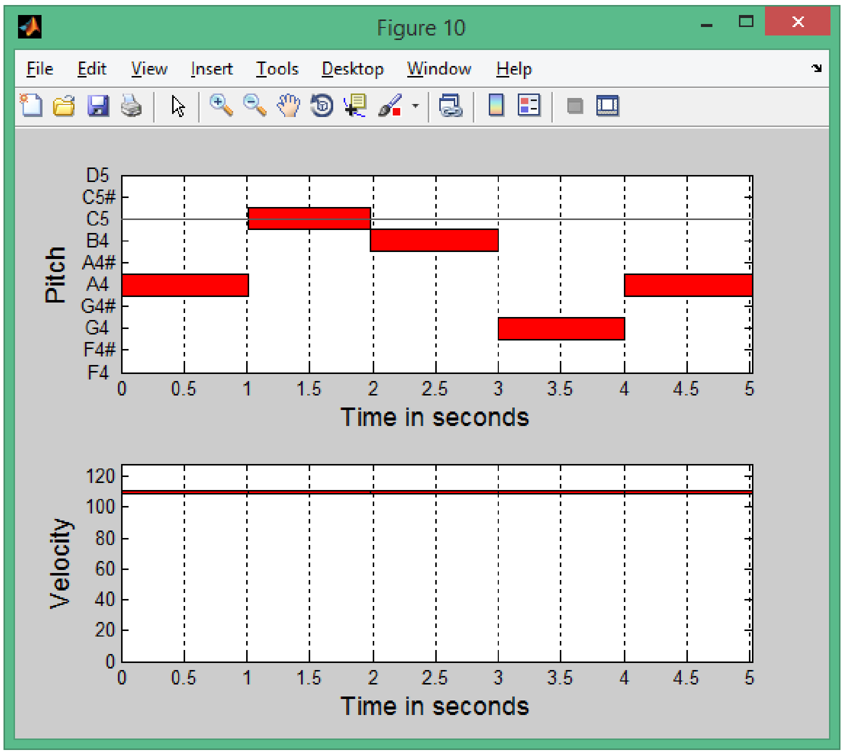


Figure 5-15: Monophonic Test Case - 1

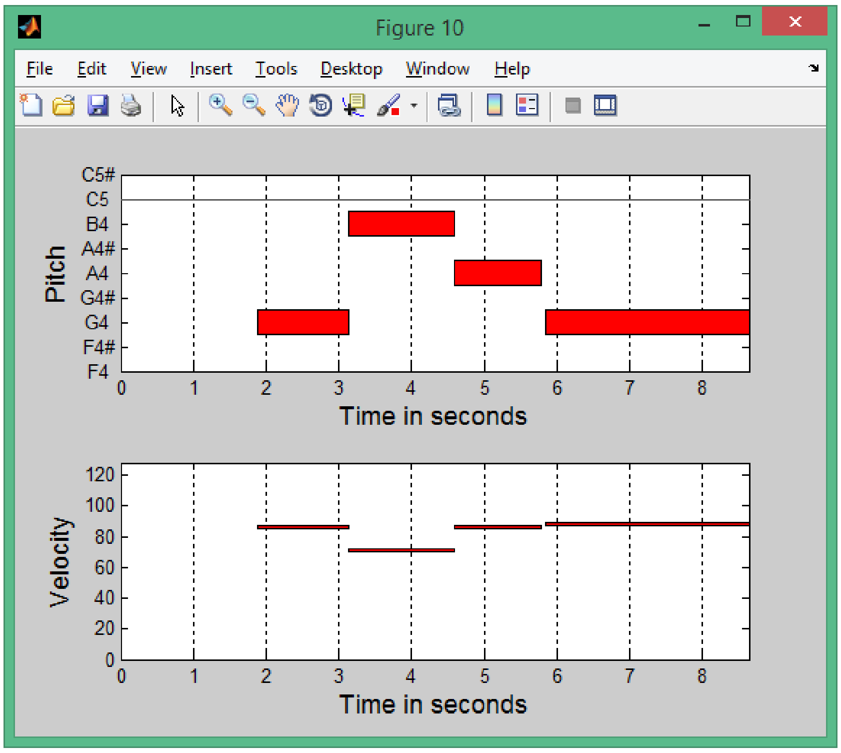


Figure 5-16: Monophonic Test Case - 2

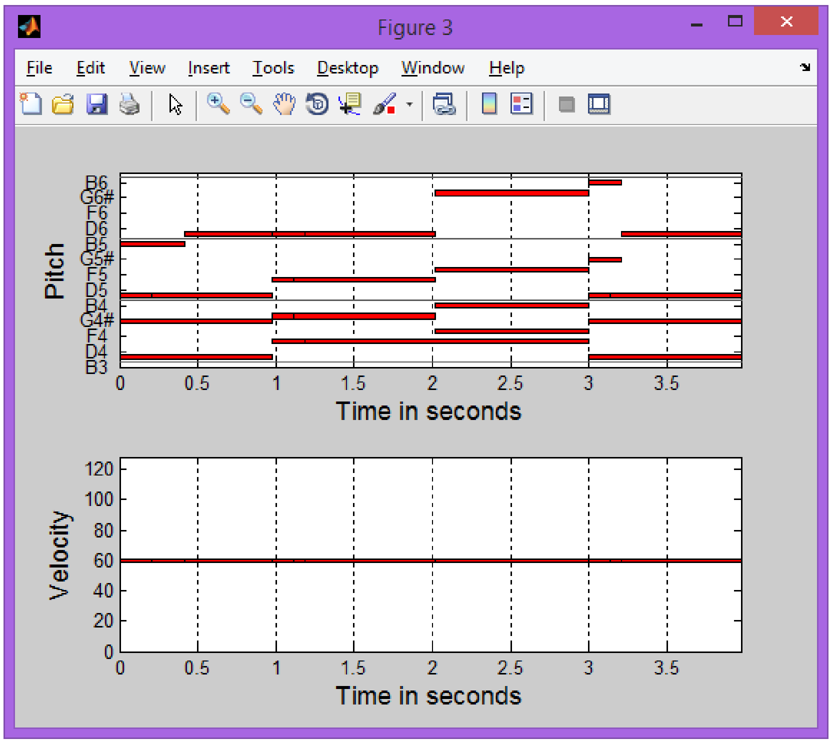


Figure 5-17: Polyphonic Test Case - 1

After testing the program on different Test Cases the results obtained are as shown in Table 5-1.

Table 5-1: Test Case Results

Test Case	**Description**	**No. of Notes Present**	**No. of Correct Notes Detected**	**Accuracy**
Mono-1	Tune played with Synthetic Sine Wave	5	5	100%
Mono-2	Tune Played with Acoustic Guitar	4	4	100%
Mono-3	Tune Played with Electric Guitar	16	15	93.75%
Poly-1	Tune Played with Piano (amount of polyphony: 2)	8	7	87.5%
Poly-2	Tune Played with Piano (amount of polyphony: 3)	9	7	77.78%

6. CONCLUSION AND FURTHER WORK

This chapter explains the concepts learned and the conclusions drawn from the project. The chapter also briefly explains the additions and modifications that can be done to the system in order for it to be used for various other activities.

6.1. Conclusion

In this project we implemented for monophonic as well as polyphonic music and the FFT based power spectrum was used for the music transcription. It was found that this technique worked satisfactorily for monophonic music and polyphonic music with amount of polyphony below 4. With increasing amount of the polyphony, the efficiency started reducing.

It was also found that when the FFT was computed, sometimes the harmonics were found to be dominant than the original note that was played. But that has been taken care of in the implementation. And on various test cases the results obtained were good. But, it can fail for the cases where the user intentionally is playing the same note in another octave consecutively. But, it should be noted here, that this case itself is very rare.

The MIDI file generated has a number of advantages on the original audio that was used as input. It was found the kind of applications suited for a MIDI file are immense and increasing day by day.

The synthesizer also provides the user with a great number of options to create different types of sounds by varying the input parameters. The sound generated was found to be satisfactory.

We have also provided users with facility to use soundfonts to generate the sound of desired instruments.

6.2. Further Work

For eliminating harmonics in the monophonic transcription Place Pitch Theory can be used. Experiments carried out by Hungarian scientist Georg von Békésy in the mid twentieth century demonstrated a simple model of how fundamental frequency detection may be carried out in the ear. Different frequencies occurring in a note stimulate different parts of the basilar membrane and this formed the basis for Place Pitch Theory. When applied to pitch detection by J. F. Schouten, he postulated that the fundamental frequency is the highest common factor of the frequencies present in a sound. For practical use in pitch detection this may be done by dividing the frequencies present in a sound by successive integers and finding the highest common factors. A table structure can be used for this as shown in Table 6-1.

Table 6-1: Place Pitch Theory example with fundamental frequency of 100 Hz

n x f0	**/ 2 (Hz)**	**/ 3 (Hz)**	**/ 4 (Hz)**	**/ 5 (Hz)**	**/ 6 (Hz)**	**/ 7 (Hz)**	**/ 8 (Hz)**	**/ 9 (Hz)**	**/ 10 (Hz)**
1	2	3	4	5	6	7	8	9	10
100	50.00	33.33	25.00	20.00	16.67	14.29	12.50	11.11	10.00
200	**100.00**	66.67	50.00	40.00	33.33	28.57	25.00	22.22	20.00
300	150.00	**100.00**	75.00	60.00	50.00	42.86	37.50	33.33	30.00
400	200.00	133.33	**100.00**	80.00	66.67	57.14	50.00	44.44	40.00
500	250.00	166.67	125.00	**100.00**	83.33	71.43	62.50	55.56	50.00
600	300.00	200.00	150.00	120.00	**100.00**	85.71	75.00	66.67	60.00
700	350.00	233.33	175.00	140.00	116.67	**100.00**	87.50	77.78	70.00
800	400.00	266.67	200.00	160.00	133.33	114.29	**100.00**	88.89	80.00
900	450.00	300.00	225.00	180.00	150.00	128.57	112.50	**100.00**	90.00
1000	500.00	333.33	250.00	200.00	166.67	142.86	125.00	111.11	**100.00**

Another technique that can be used for proper windowing of the audio signal can be Note onset detection [16]. Note onset detection and deciding whether the subsequently detected frequencies represent the fundamental frequencies of a note or harmonics of a note are the two main tasks in music transcription. Using the onset detection system, the onsets and the offsets for each note or chord can be identified. Therefore the musical signal is broken up into windows, according to the onsets and offsets of the notes or chords. These windows are then analyzed to see what note or notes were played in a specific window of information.

As the number of notes increases the accuracy of the polyphonic note recognition decreases for that the note masking system can be used. In the note masking system, a mask can be created for each of the note which is played on specific instrument. After detection of a note the mask corresponding to that note can be applied so that it suppresses the detected note and other notes can be recognized [16]. This approach will increase the efficiency in the polyphonic notes recognition but this method can fail if the sound of instrument significantly vary from the masks.

This system can be further extended to Musical Score Generation which will generate the sheet music of played music which can be used as a record of, a guide to, or a means to perform, a piece of music by other musicians.

This system can also be extended to recognize the Indian Classical Ragas or Western Music by studying the pattern in the recognized notes.

The VSTi (Virtual Studio Technology instrument) plugin of this system can be implemented in the programming language like C++ which will extend the functionality of system like audio and MIDI output to Digital Audio Workstation (DAW) [17]. Virtual Studio Technology (VST) is a software interface that integrates software audio synthesizer and effect plugins with audio editors and hard-disk recording systems.

REFERENCES

[1] Schmidt-Jones, C. (2013) “What Kind of Music is That?” [online], Available: http://cnx.org/content/m11421/latest. [Accessed: Jan. 17, 2014].

[2] The Colombia Encyclopedia, Sixth Edition (2013), “Arabian music” [online], Available: http://www.encyclopedia.com/doc/1E1-Arabianm.html. [Accessed: Jan. 17, 2014].

[3] Encyclopedia Britannica (2012), “Octave” [online], Available: http://www.britannica.com/EBchecked/topic/424821/octave. [Accessed: Jan. 17, 2014].

[4] Scott Wilson (2003), “WAVE PCM soundfile format” [online], Available: https://ccrma.stanford.edu/courses/422/projects/WaveFormat/. [Accessed: Jan. 25, 2014].

[5] The Sonic Spot (2007), "Wave File Format" [online], Available: http://www.sonicspot.com/guide/wavefiles.html. [Accessed: Jan. 25, 2014].

[6] Center for Computer Assisted Research in the Humanities at Stanford University: Music 253, “Outline of the Standard MIDI File Structure” [online], Available: http://www.ccarh.org/courses/253/handout/smf/. [Accessed: Jan. 25, 2014].

[7] Dave Marshall (2001), “MIDI Messages” [online], Available: http://www.cs.cf.ac.uk/Dave/Multimedia/node158.html. [Accessed: Jan. 25, 2014].

[8] Creative Worldwide Support (2013), “SoundFonts - General Information and Essential Troubleshooting” [online], Available: http://support.creative.com/kb/ShowArticle.aspx?sid=5184. [Accessed: Jan. 25, 2014].

[9] Tao Li, Ogihara, M., “Toward intelligent music information retrieval,” IEEE Transactions on Multimedia, Vol. 8, June 2006, pp. 564 – 574.

[10] A. Taylan Cemgil, Hilbert J. Kappen and David Barber, “A Generative Model for Music Transcription,” IEEE Transactions on Audio, Speech, and Language Processing, Vol. 14, No. 2, March 2006, pp. 679-694.

[11] Youssef, K., Peng-Yung Woo, “Music Note Recognition Based on Neural Networks,” Natural ICNC '08. Fourth International Conference on Computation, Vol. 2, 2008, pp. 474 - 478.

[12] Runfeng Yang, JianyongBian and LipingXiong, “Frequency to MIDI Converter for Musical Instrument Microphone System,” Consumer Electronics, Communications and Networks (CECNet), 2012, pp. 2597-2599.

[13] Say Wei Foo, Wei Thai Lee, “Recognition of piano notes with the aid of FRM filters,” First International Symposium on Control, Communications and Signal Processing, 2004, pp. 409 - 413.

[14] Dimitrios Fragoulis, Constantin Papaodysseus, Mihalis Exarhos, George Roussopoulos, Thanasis Panagopoulos, and Dimitrios Kamarotos, “Automated Classification of Piano–Guitar Notes,” IEEE Transactions on Audio, Speech, and Language Processing, Vol. 14, No. 3, May 2006, pp. 1040-1050.

[15] Barros, J., Diego, R.I., "On the use of the Hanning window for harmonic analysis in the standard framework," IEEE Transactions on Power Delivery, Vol. 21, No.1, 2006, pp. 538 - 539.

[16] Ronan Kelly, “Automatic Transcription of Polyphonic Music Using A Note Masking Technique,” Masters of Engineering, University of Limerick, March 2010.

[17] Ann Franchesca B. Laguna, Nicanor Marco P. Valdez and Rowena Cristina L. Guevara, “MIDI Implementation of a Kulintang Modal Synthesizer using the VST2.4 Standard,” TENCON 2012 - 2012 IEEE Region 10 Conference, 2012.

ACKNOWLEDGEMENT

We whole-heartedly would like to thank our project guide, Dr. Archana Patankar, Associate Professor, Computer Engineering Department, Thadomal Shahani Engineering College. Without her help and guidance throughout, this project wouldn’t have been possible. She has been supportive and has always shown a keen interest in our work on the project. Her inputs on how we should go about with the things, has been valuable. Every time that we were confused about anything, she was there to show us a way out and we once again thank her for all of it.

We would also like to extend our gratitude to our H.O.D., Mr. Jayant Gadge for providing us with good facilities in the college and the entire staff of Computer Engineering Department, for being cooperative and helpful in every way that they could.

Murtaza Dhuliawala

Sanjay Khatwani

Hirak Modi

SUMMARY

An audio to MIDI converter was successfully implemented in Matlab. The system used FFT based power spectrum for music transcription of monophonic as well as polyphonic music and generated good results for a sufficient number of test cases. After the information about a note, its duration and velocity were retrieved from an input audio file, its equivalent MIDI file was generated. This MIDI file is comparatively very small in size and has many advantages. This MIDI file can also be given to a synthesizer to generate any desired tone irrespective of the instrument or the tone of the original music piece. In the system, additive and FM synthesizers were successfully implemented along with a wide range of customization options provided to the user. Another sound generation module has been implemented using soundfonts. This takes as input the MIDI file previously generated and produces the sound of a desired instrument, which would be independent of the original instrument on which a music was played. User is given options to select the instrument as per their need. Soundfonts were successfully used to implement this functionality.