Mini Project Report on

Detecting a particular frequency/note of a musical instrument

Avinash Kumar (16EE153)

Gayatri Indukumar (16EE218)

Megh Manoj Bhalerao (16EE234)

Under the Guidance of,

Dr. Krishnan CMC

Department of Electrical and Electronics Engineering, NITK Surathkal

Date of Submission: 03-17-2019

in partial fulfillment for the award of the degree

of

Bachelor of Technology

In

Electrical and Electronics Engineering

At

Department of Electrical and Electronics Engineering

National Institute of Technology Karnataka, Surathkal

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

Department of Electrical and Electronics Engineering

CERTIFICATE

This is certify that the report entitled **'Detection of a particular frequency/note of a musical instrument',** submitted by **Avinash Kumar (16EE153), Gayatri Indukumar (16EE218) and Megh Manoj Bhalerao (16EE234)** is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of B.Tech in Electrical and Electronics Engineering at National Institute of Technology Karnataka, Surathkal.

Dr. Krishnan CMC

Assistant Professor
Department of EEE
NITK, Surathkal

ACKNOWLEDGEMENT

We would like to thank **Dr. Krishnan C M C** for giving us the opportunity to be a part of this wonderful learning experience by the means of this project. His willingness to give his precious time so generously has been very much appreciated by all of us. We would also like to thank **Sai Chaitanya** and **ISTE** for providing us with necessary components for the implementation of the project. Finally we wish to thank the **Department of electrical and electronics engineering, NITK** for providing us with courses and projects which help us grow as engineers.

Abstract

Frequency Detection is relatively simple on MATLAB and as a mathematical concept, but when it comes to hardware implementation, it poses a number of difficulties. Hardware implementation of frequency detection algorithms poses a vast scope in terms of building accurate Guitar/Piano Tuners, Vibration Analyzers, Heartrate Monitors, MEMs Sensor Analysis and Laboratory Instruments.
In this project, we explore frequency detection using an Arduino. Particularly detection of frequency from musical notes produced by an instrument. There are a number of approaches available for this, we explore two methods here, namely Fast Fourier Transform and Auto-correlation.

Keywords: *FFT, Auto-correlation*

Contents

List of Figures

1 Introduction

Detection of an audio parameter on a microprocessor is an ample researched topic, Mahmood et al[1] implement a frequency and amplitude detector for sinusoidal signals with information only from three successive samples of the input signal. Alam et al[2] use a PIC microcontroller to measure power frequency, where the input is passed through a zero crossing detector (ZCD) and converted into a square wave which is further processed to obtain the output. Work relating to this area of research has a wide application in power systems, automatic gain and frequency control etc.

The project essentially revolves around implementation of the frequency detection technique in the microprocessor (Arduino Uno is used here). There are two main stream ways to implement this task on arduino, use of a Fast Fourier Transform (fft) and Auto-correlation. Here, in our work we use both of these mentioned algorithm and implement circuits to overcome the limitations of an arduino in processing a real-time audio signal.The specifications of the circuit and the code are described in section 2 and 3 respectively.

2 Circuit

2.1 Pertaining to the FFT algorithm

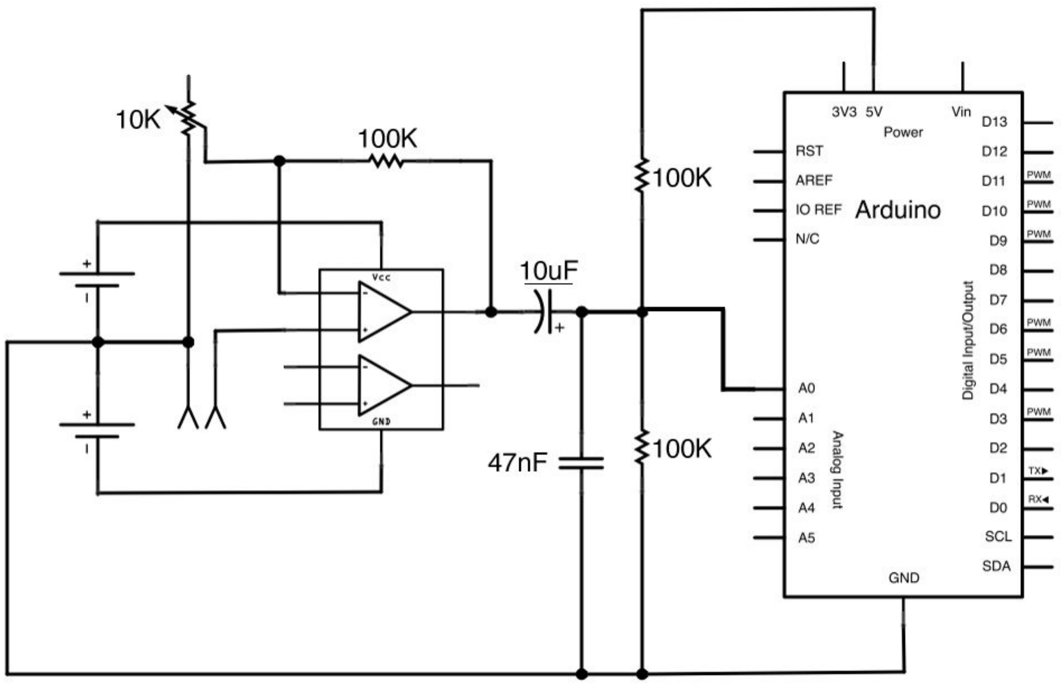


Figure 2.1: Connection diagram of the complete setup

The circuit shown above constitutes the complete setup of the project. The circuit can be sub-divided into three main components depending upon its functionality : 1) Taking Audio input 2) Amplification and 3) Adding DC offset. Audio input is given to the Arduino with the help of an audio-jack. In our case we use a Application[3] to generate a mono-tone which is given to the circuit via a mono audio jack.This signal is amplified with the use of a non-inverting amplifier configuration using the TL082 IC. The amplifier basically performs two functions, first it increases amplitude from + or - 200mV to + or - 2.5V (ideally). Secondly, as the audio source is connected only to the IC, any loading in the rest of the circuit will not affect the audio source. Hence, isolating the audio source. The gain of the non-inverting amplifier is given by :

$$V_o = (1 + \frac{R2}{R1}) * V_{in} \tag{1}$$

where Vo is the output signal/voltage, Vin is the input signal, R2 is the feedback resistance (100K ohm in our case) and R1 is the resistance attached to the inverting pin of the Op-Amp (10K ohm potentiometer in our case). The arduino can take an input only in the range of (0-5)V implying the need for a DC offset to be added to the signal. If this is not performed then the portion of the signal below 0V or negative values present in the signal will get clipped. Following this operation, the signal is given to the arduino for processing. This input circuit is inspired from the work of amandaghassaei[4].

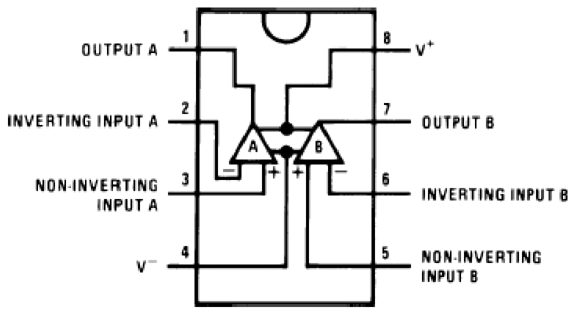


Figure 2.2: Pin diagram of Tl082 IC

2.2 Pertaining to the auto-correlation algorithm

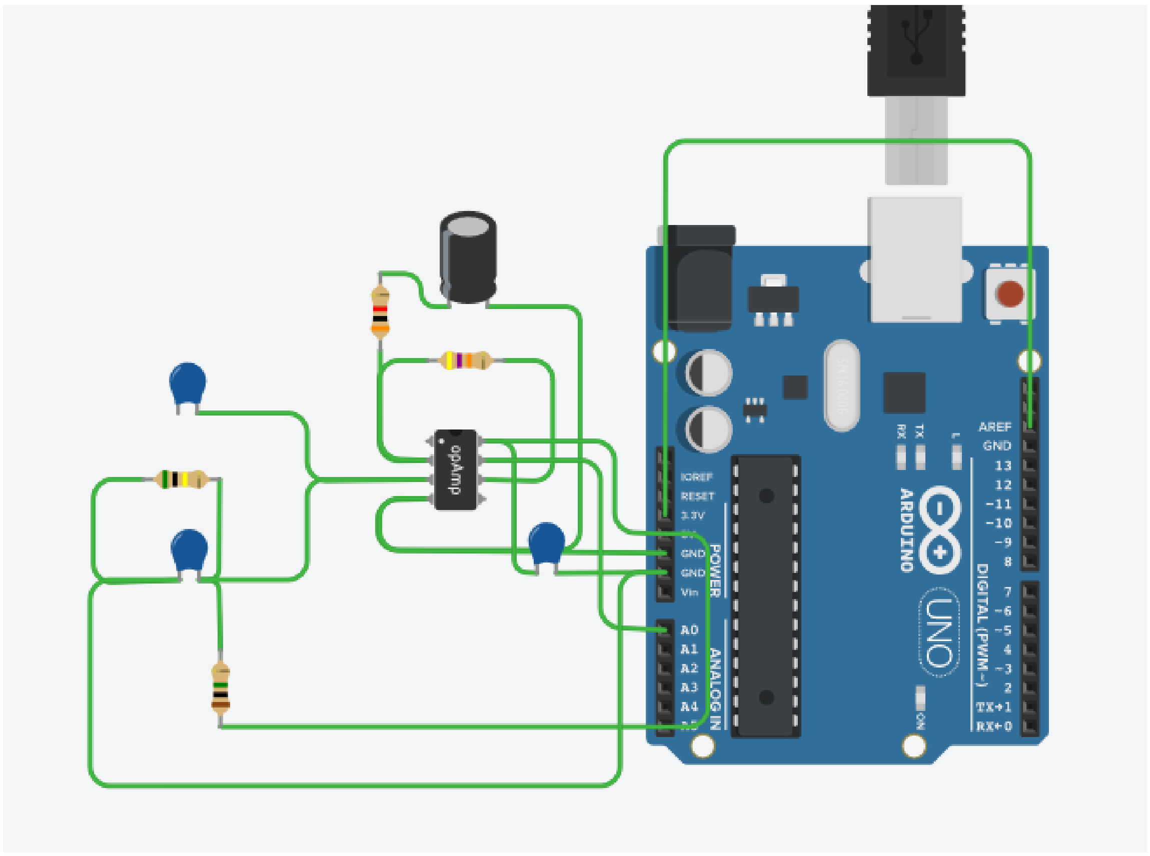


Figure 2.3: Schematic of the complete setup

This diagram is built on **Tinkercad** and is a schematic of the complete setup. The circuit contains a 741 IC (Op-Amp), the input from the audio jack (given to the free end of the capacitance on the upper left corner) is amplified via the IC and an offset is added due to the circuit setup. The range of the output is of a maximum of ± 3.3 V. Hence, for better precision, the AREF pin of the arduino is given to the 3.3V port on the Arduino board. This setup was tried extensively with the code specified in the next section but was unable to give legible results, even though the auto-correlation algorithm is known to perform better at frequency identification in the literature.

3 Arduino Code

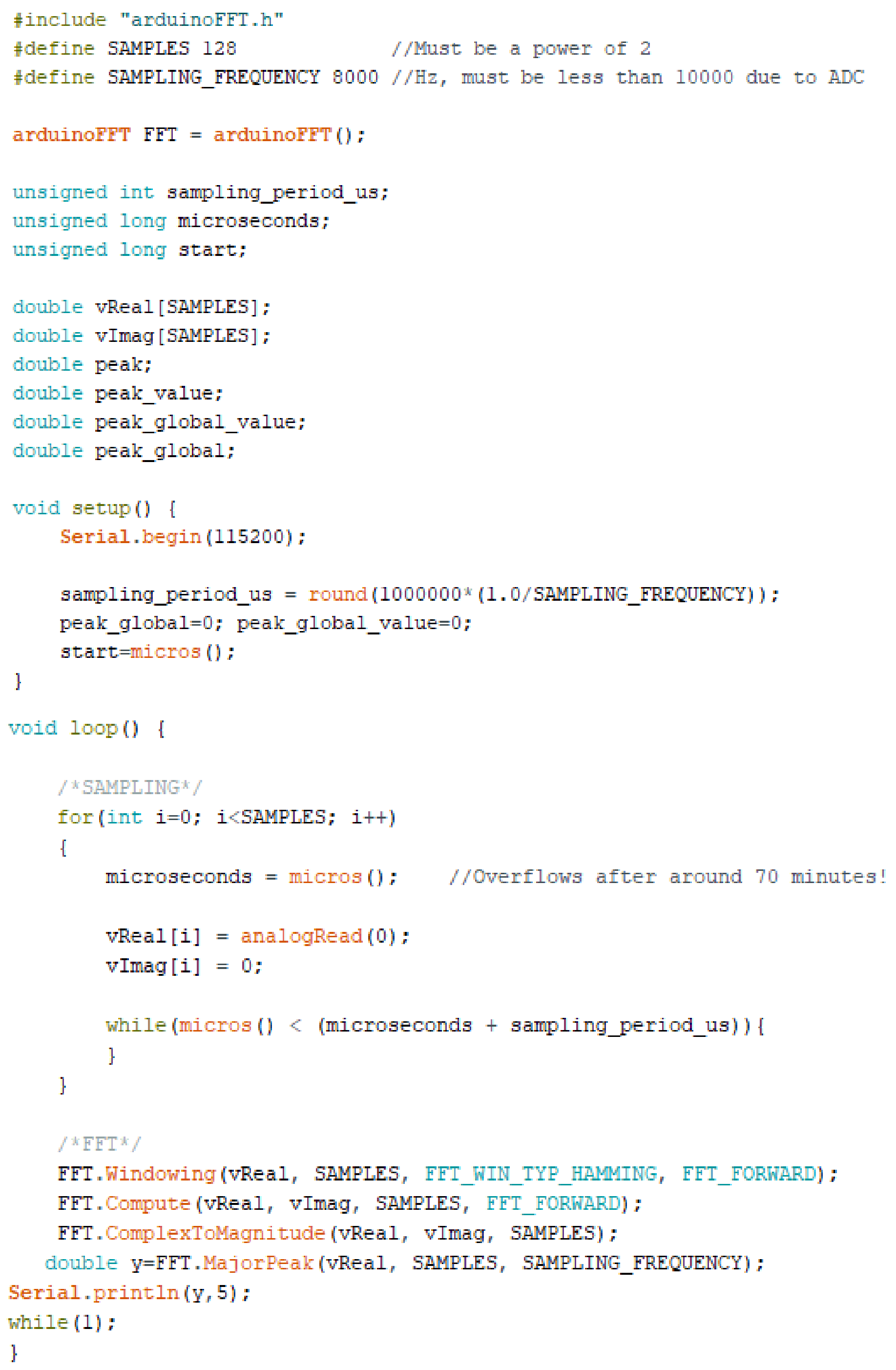
CODE 1:The above code uses arduinoFFT library which performs a floating-point Fast Fourier Transform on a sampled signal. The for loop collects samples (here 128 samples at a time) at the pre-defined sampling frequency, using the analogRead() function which reads input signals that are connected to the analog pins of an Arduino. The while loop ensures that the next sample is collected after 1/Fs seconds. Thus, Sampling the input at Fs Hz. The Sampled signal is then passed through a Hamming window to avoid Spectral leakage using FFT.Windowing(). Then its FFT is calculated and the major peaks are printed out as the output. The peaks can then be visualized via the Serial Plotter.

CODE 2: Auto-correlation is the process of comparing a segment of a signal with its delayed version. Thus, we map the similarities between 2 segments of a signal as a function of the delay. This method can be used to obtain the component frequencies in a given signal. The code describes the extraction of component frequencies from a noisy signal using auto-correlation. The initial for loop performs the auto-correlation by comparing the segments of the input signal for a varying delay of i. 128 is subtracted from each value because analogRead() values are 8bit unsigned and we require signed values. To detect the location of the first peak after the maximum we use a simple peak detector coded as a State Machine. The state machine moves from one state to the next when an event occurs as follows:

STATE0: Set thresh the threshold under which value well ignore the data: **NEW STATE** = 1

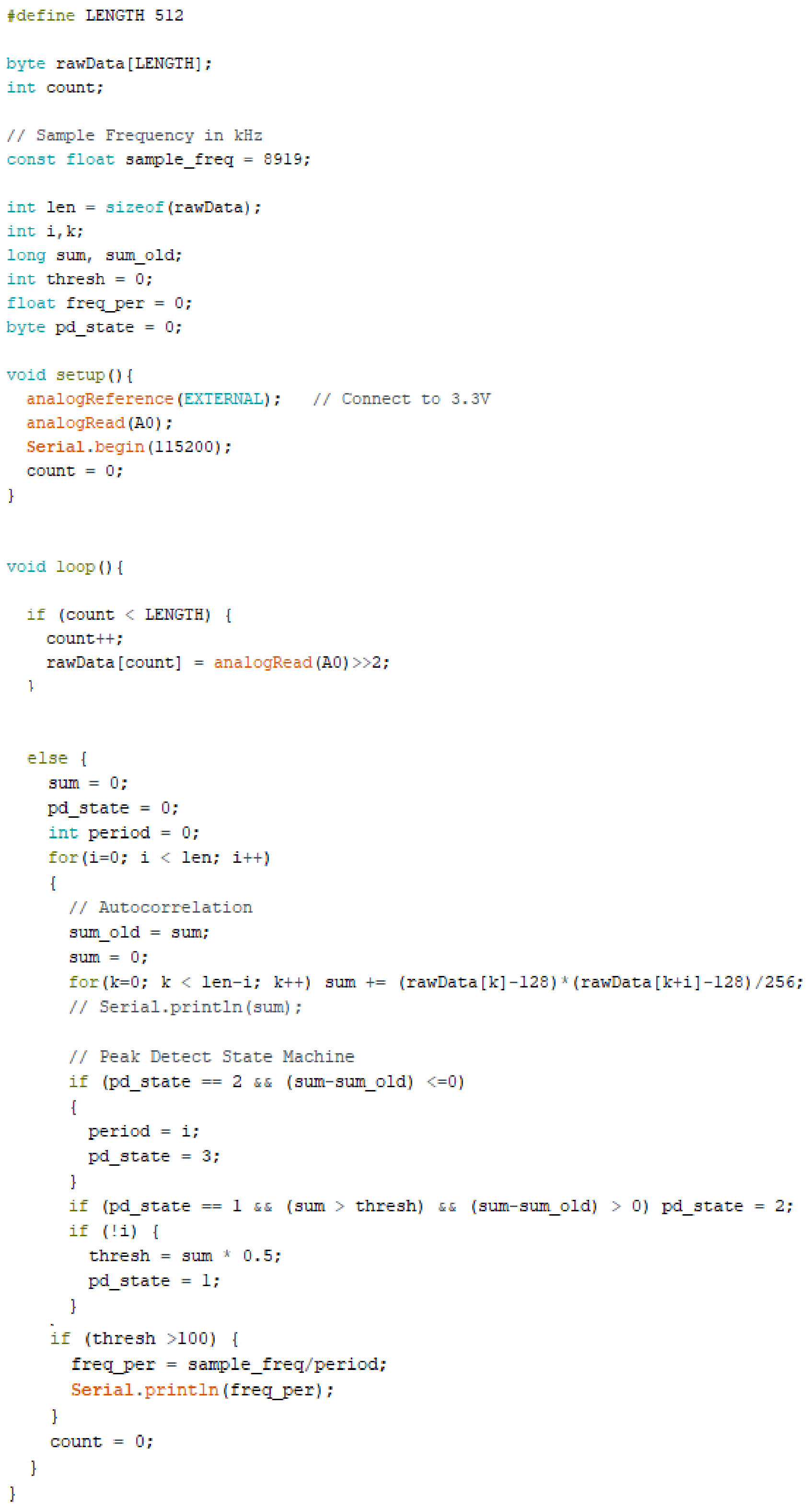
STATE1: look for the signal being above the threshold AND the slope of the signal is positive: **NEW STATE** = 2

STATE2: look for the slope of the signal is negative or zero. If so, weve found the PEAK!: **NEW STATE** = 3 This code is inspired from the work of Akellyril[5].



```
#include "arduinoFFT.h"
#define SAMPLES 128             //Must be a power of 2
#define SAMPLING_FREQUENCY 8000 //Hz, must be less than 10000 due to ADC

arduinoFFT FFT = arduinoFFT();

unsigned int sampling_period_us;
unsigned long microseconds;
unsigned long start;

double vReal[SAMPLES];
double vImag[SAMPLES];
double peak;
double peak_value;
double peak_global_value;
double peak_global;

void setup() {
    Serial.begin(115200);

    sampling_period_us = round(1000000*(1.0/SAMPLING_FREQUENCY));
    peak_global=0; peak_global_value=0;
    start=micros();
}

void loop() {

    /*SAMPLING*/
    for(int i=0; i<SAMPLES; i++)
    {
        microseconds = micros();    //Overflows after around 70 minutes!

        vReal[i] = analogRead(0);
        vImag[i] = 0;

        while(micros() < (microseconds + sampling_period_us)){
        }
    }

    /*FFT*/
    FFT.Windowing(vReal, SAMPLES, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
    FFT.Compute(vReal, vImag, SAMPLES, FFT_FORWARD);
    FFT.ComplexToMagnitude(vReal, vImag, SAMPLES);
   double y=FFT.MajorPeak(vReal, SAMPLES, SAMPLING_FREQUENCY);
Serial.println(y,5);
while(1);
}
```

Figure 3.1: Code-1 : Pertaining to the fft Algorithm



```
#define LENGTH 512

byte rawData[LENGTH];
int count;

// Sample Frequency in kHz
const float sample_freq = 8919;

int len = sizeof(rawData);
int i,k;
long sum, sum_old;
int thresh = 0;
float freq_per = 0;
byte pd_state = 0;

void setup(){
  analogReference(EXTERNAL);   // Connect to 3.3V
  analogRead(A0);
  Serial.begin(115200);
  count = 0;
}


void loop(){

  if (count < LENGTH) {
    count++;
    rawData[count] = analogRead(A0)>>2;
  }


  else {
    sum = 0;
    pd_state = 0;
    int period = 0;
    for(i=0; i < len; i++)
    {
      // Autocorrelation
      sum_old = sum;
      sum = 0;
      for(k=0; k < len-i; k++) sum += (rawData[k]-128)*(rawData[k+i]-128)/256;
      // Serial.println(sum);

      // Peak Detect State Machine
      if (pd_state == 2 && (sum-sum_old) <=0)
      {
        period = i;
        pd_state = 3;
      }
      if (pd_state == 1 && (sum > thresh) && (sum-sum_old) > 0) pd_state = 2;
      if (!i) {
        thresh = sum * 0.5;
        pd_state = 1;
      }
    }
    if (thresh >100) {
      freq_per = sample_freq/period;
      Serial.println(freq_per);
    }
    count = 0;
  }
}
```

Figure 3.2: Code-2 : Pertaining to the Auto-correlation Algorithm

4 Implementation

Based on performance of the above two codes for their circuit setups described in section 2. The fft algorithm is implemented. Following are a few graphical illustrations for the same.

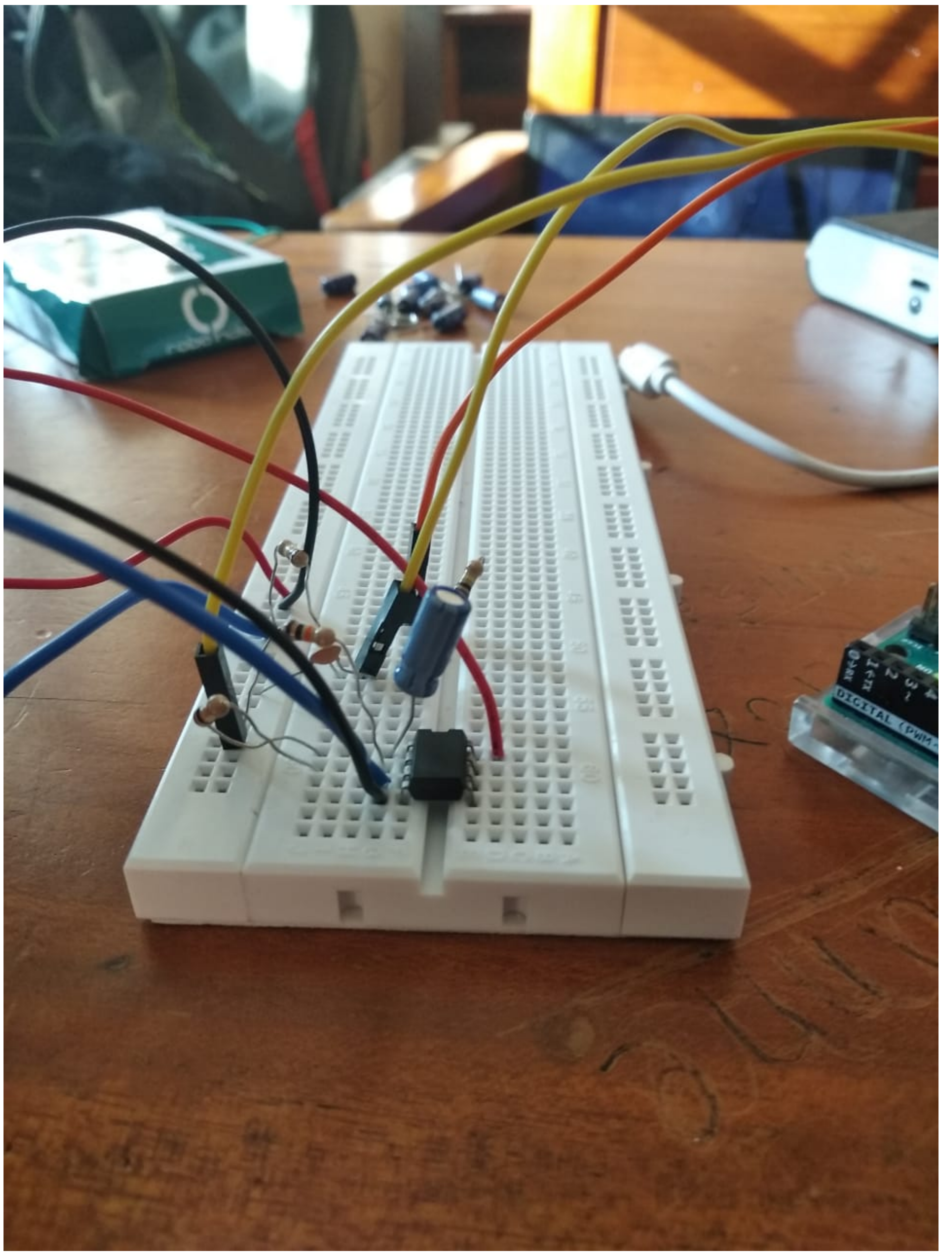


Figure 4.1: Circuit Setup in Real time

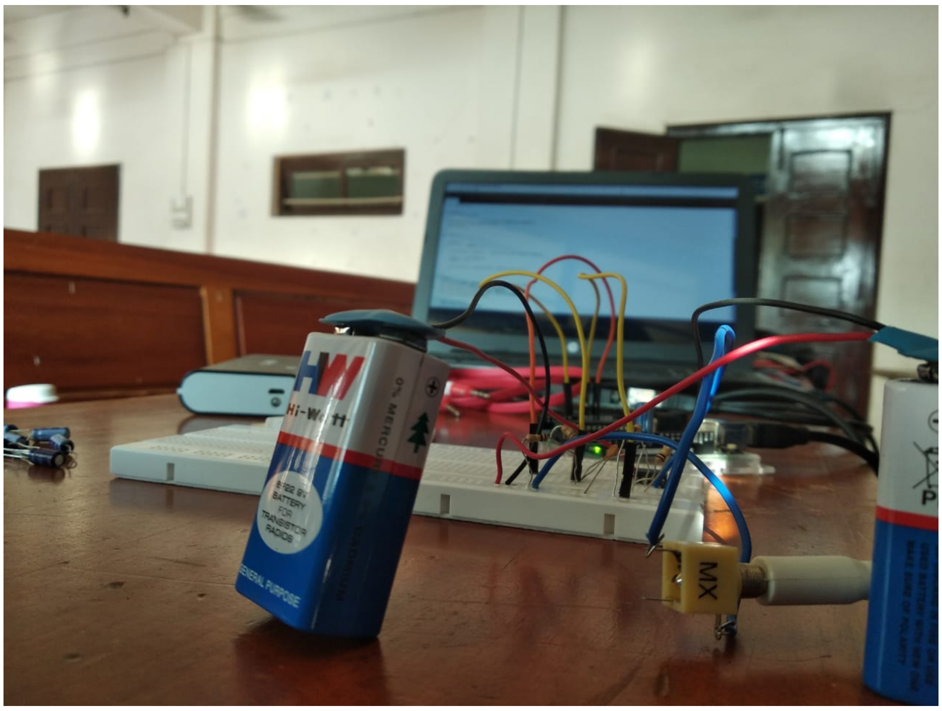


Figure 4.2: Overview of the complete setup

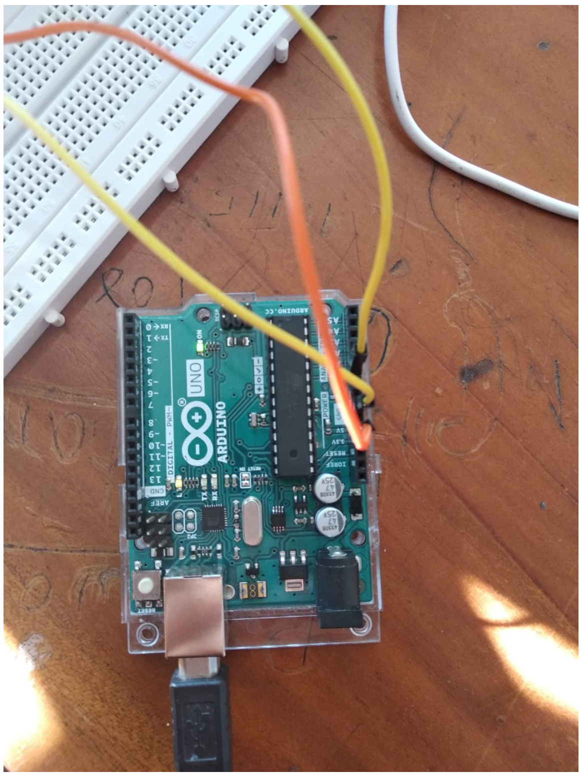
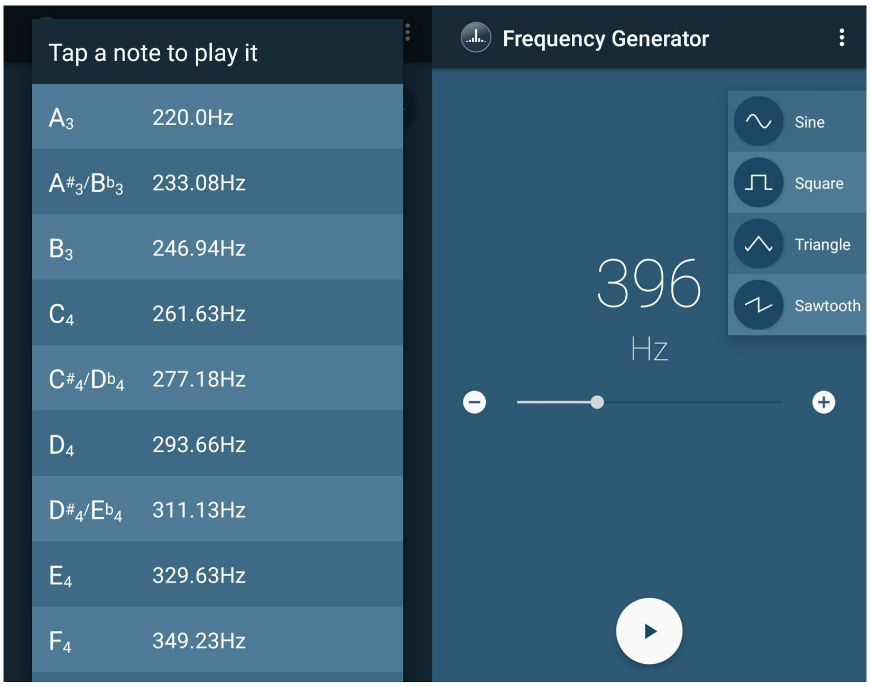


Figure 4.3: Arduino Connections



Tap a note to play it

A_3	220.0Hz
$A\#_3/Bb_3$	233.08Hz
B_3	246.94Hz
C_4	261.63Hz
$C\#_4/Db_4$	277.18Hz
D_4	293.66Hz
$D\#_4/Eb_4$	311.13Hz
E_4	329.63Hz
F_4	349.23Hz

Frequency Generator

396 Hz

Figure 4.4: Application used to give audio input to the system setup

5 Problems faced

One of the greatest limitations of using an Arduino involves reading the signal using analogRead(). The Arduino microcontroller (commonly an Atmega328 integrated circuit) is capable to convert Analog readings into a digital value through its internal ADC (Analog to Digital Converter). This conversion is never perfect and its precision is determined by the ADC resolution, in our case being 10 bits: we get a value between 0 and 1023 representing usually a voltage between 0V and 5V. This means our readings have a resolution of about 5mV and we will not be able to determine the difference between 1.000V and 1.004V. Also, the Arduino has only one ADC to convert signals in multiple Analog input pins. This condition limits the speed of the analogRead() function and, in conjunction with the circuit capacitance and impedance, can lead to wrong readings. This proves to be a problem if we require accurate high-speed readings. It is also necessary to provide an appropriate voltage at AREF pin of the Arduino. All the values returned by the ADC are determined by comparing the internal capacitor voltage to a reference voltage.

6 Results and Conclusions

Upon implementation of the above mentioned circuit and code (pertaining to the fft algorithm), we get the arduino output to be about ± 10 percent of the actual mono-tone or frequency being tested. This was rigorously tested to make sure that there exists a mapping between the input and the output. We find that upon playing a mono tone consistently, the arduino output range lies at almost a particular frequency indicating stability but with a certain degree of error. For frequencies above a certain range, there can be issues while implementing the project on arduino as the sampling frequency in an arduino is limited to 10000Hz due to the limitations of the AD converter present. The design task can be implemented via many techniques, fft being one of them. Certainly, the performance of this setup can be enhanced with different approaches.

7 Future work

In the present circuit as of now, there is no presence of an external display device except the serial monitor present in the arduino platform. The goal for this to implement a display using a screen etc. We will try to minimize the error present in the output of the setup. Also we will try to implement auto-correlation rather than fft as it is known to give better outputs.

References

[1] Mahir K. Mahmood, Janan E. Allos and Majid A. H. Abdul-Karim ”Microprocessor Implementation of a Fast and Simultaneous Amplitude and Frequency Detector for Sinusoidal Signals”, IEEE Transactions on Instrumentation and Measurement (Volume: IM-34 , Issue: 3 , Sept. 1985)

[2] Khairul Alam, Tanmoy Chakraborty, Srabana Pramanik (Chaudhury), Debabrata Sarddar, Satadal Mal ”Measurement of Power Frequency with Higher Accuracy using PIC Microcontroller”

[3] Hoel Boedec’s Android application ”Frequency Sound generator” available on google playstore.

[4] `https://www.instructables.com/id/Arduino-Audio-Input/`by Ammanda Ghassaei

[5] `https://github.com/akellyirl/Arduino-Guitar-Tuner` by Akellyril