Male-Female Voice Recognition

17uec064 – Keshav Sikawat

17uec065 – Khyati Goyal

17uec066 – Kshish Ashish

17uec068 – Kurapapu Khagesh Chakri

AIM -

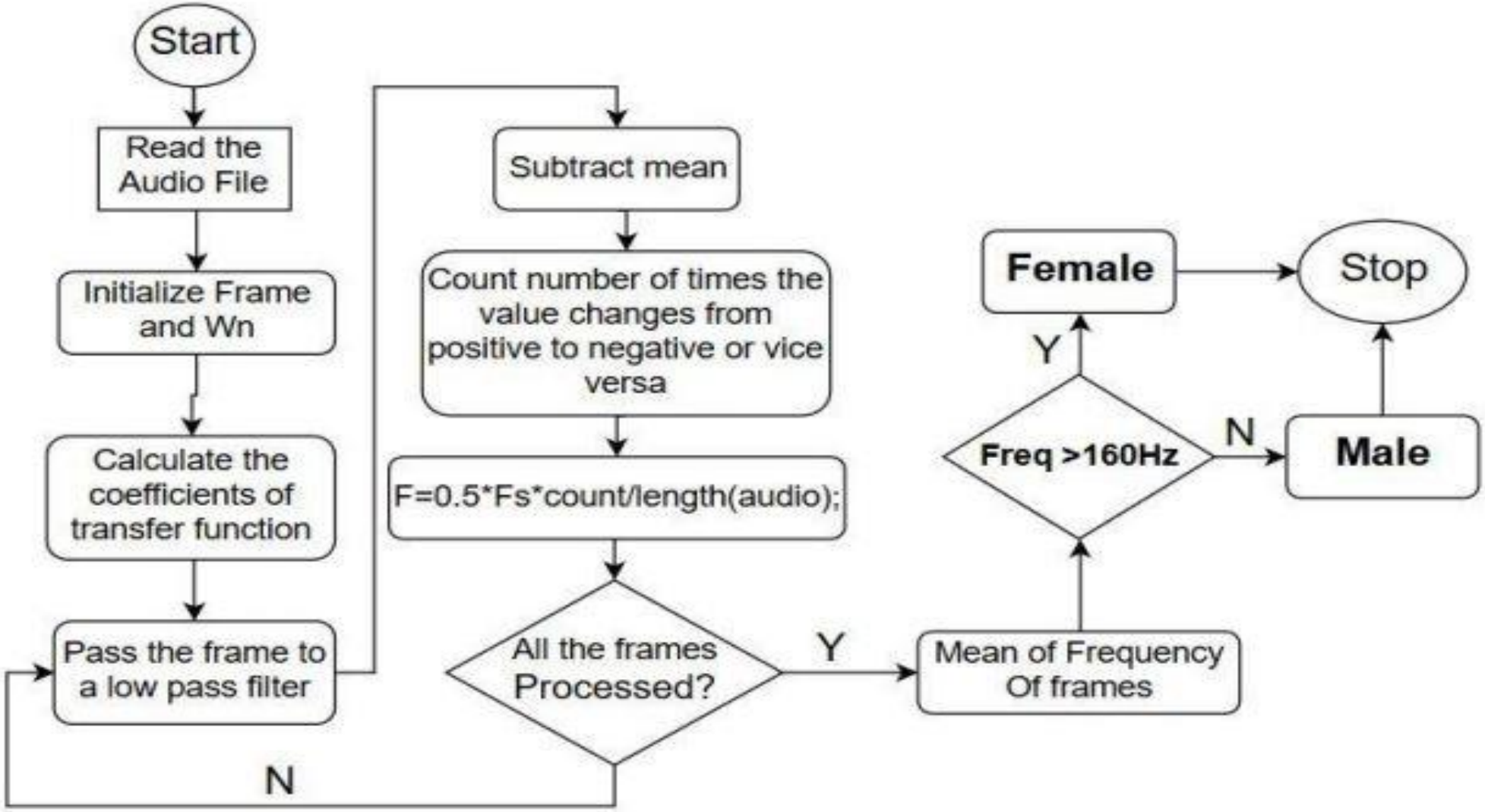
A voice sample is given as an input to the program & it should report whether it’s a male voice or a female voice.

APPROACH -

Male and female voices have different frequencies. Typical frequency for a male ranges from 85Hz to 180Hz while that for female ranges from 165Hz to 255Hz. For identification, the sample can be passed from high pass and low pass filter or band pass filter according to frequency.

We built a MATLAB program which returns the fundamental frequency, F_0 of any input audio file with samples F_s. Thus, we can directly compare it with 165 Hz and identify whether the voice is of a male or a female.

Flowchart of the program



MATLAB Code snippets

```
clc;
clear all;
close all;

[voice ,Fs]=audioread('f2.wav'); % Read the audio sample
frame=3500; % Set the frame rate

[b0,a0]=mybutter(350/(Fs/2)); % Get the coefficient of the filter matrix

% Identify the frequency of each frame
for i=1:length(voice)/frame
   x=voice(1+(i-1)*frame:i*frame);

   inputframe = abs(x);
   inputframe=myfilter(b0,a0,inputframe);
   inputframe = inputframe-mean(inputframe);

   nextframe=zeros(length(inputframe),1);
   nextframe(1:length(x)-1)=inputframe(2:length(x));
   count=length(find((inputframe>0 & nextframe<0) | (inputframe<0 & nextframe>0)));
   F0(i)=0.5*Fs*count/length(x);
end
```

```
Fx=mean(F0); % Take mean of all the frequency for each frame

% Display the output frequency
fprintf('Estimated frequency is %3.2f Hz.\n',Fx);

% Display the final Gender
if Fx>165   % set the threshold
   fprintf('Female Voice\n');
else
   fprintf('Male Voice\n');
end
```

MATLAB Code snippets

```
function filtered = myfilter(b, a, raw)

filtered = zeros(size(raw));
    for n = 3:size(raw,1)
        filtered(n,1) = b(1)* raw(n,1)    + b(2)* raw(n-1,1) + b(3)* raw(n-2,1) ...
                      - a(1)*filtered(n,1)    - a(2)*filtered(n-1,1) - a(3)*filtered(n-2,1) ;
    end
```

Low Pass Filter -
There are 'b' numerator & 'a' denominator coefficients of the rational transfer function, specified as a vector. Raw is the input data file that we are traversing column wise. We used the formula

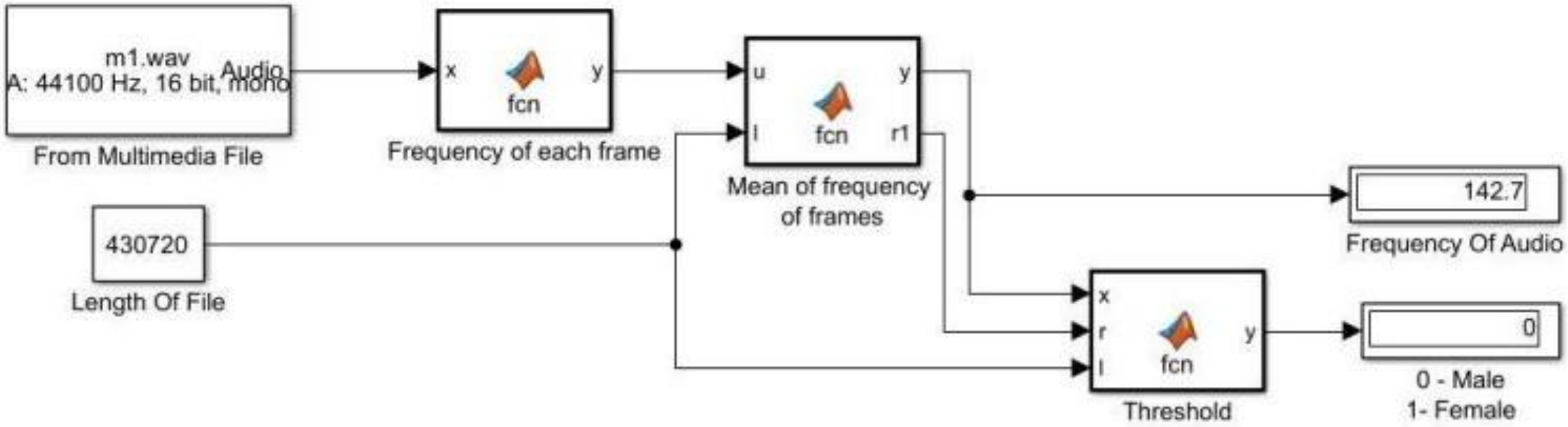
a(1)y(n) = b(1)x(n) + b(2)x(n-1) +…+ b(nb+1)x(n-nb) – a(2)y(n-1) -…- a(na+1)y(n-na) and a(1)=1.
And the transfer function used is:

$$Y(z) = \frac{b(1) + b(2)z^{-1} + \ldots + b(n_b + 1)z^{-n_b}}{1 + a(2)z^{-1} + \ldots + a(n_a + 1)z^{-n_a}} X(z),$$

```
function [b,a] = mybutter(fc)

V  = tan(fc *pi/2);
Sg = V ^ 2;
factorized = V * [-1-1i, -1+1i] / sqrt(2);

%all pass filter
P = (1 + factorized) ./ (1 - factorized);
Gain = real(Sg / prod(1 - factorized));

b = Gain * [1, 2, 1];
a = real(poly(P));
end
```

Butterworth Filter -
We have created a 2nd order Butterworth filter. "poly()" function is used to build polynomials of given roots. Derivation eq. used is: $s^2 + (\sqrt{2})s + 1$

Simulink model snippet



C coding snippets

Low Pass Filter -

```
float filter(float cutofFreq){
    float RC = 1.0/(cutofFreq * 2 * M_PI);
    float dt = 1.0/SAMPLE_RATE;
    float alpha = dt/(RC+dt);

    return alpha;
}

float filteredArray[numSamples];
filteredArray[0] = data.recordedSamples[0];

for(i=1; i<numSamples; i++){
    if(i%SAMPLE_RATE == 0){
        cutoff = cutoff - 400;
    }
```

C coding snippets

Butterworth Filter -

```
DoubleComplexVector ButterworthFilter( DoubleComplexVector signal,
   double sampleFrequency, int order, double f0, double DCGain )
    {

      var N = signal.Length;

      // Apply forward FFT
      var fft = new DoubleComplexForward1DFFT( N );
      var signalFFT = fft.FFT( signal );

      if ( f0 > 0 )
      {

        var numBins = N / 2;  // Half the length of the FFT by symmetry
        var binWidth = sampleFrequency / N; // Hz
```

```
// Filter
        System.Threading.Tasks.Parallel.For( 1, N / 2, i =>
        {
          var binFreq = binWidth * i;
          var gain = DCGain / ( Math.Sqrt( ( 1 +
                      Math.Pow( binFreq / f0, 2.0 * order ) ) ) );
          signalFFT[i] *= gain;
          signalFFT[N - i] *= gain;
        } );

      }

      // Reverse filtered signal
      var ifft = new DoubleComplexBackward1DFFT( N );
      ifft.SetScaleFactorByLength(); // Needed to get the correct amplitude
      signal = ifft.FFT( signalFFT );

      return signal;
    }
```

Conclusion

- We create a function of our own & identify the voice sample. Initially, we tried to use inbuilt function “pitch” to find frequency of the entire audio file.
- But due to noise in the sample, we had to divide the sample into smaller frames, remove noise using our low pass filter, then calculate the deviation from mean in each frame & it’s mean.
- This method identifies the voice sample with an accuracy rate of 96%. Out of the 50 audio samples we used, 48 were detected correctly. This rate can be improved by using higher order filters.
- Since there is an overlap of frequencies from 165Hz to 180Hz, using mean proves as a much better approach as the max. frequency in male voice may go upto 180Hz while the mean frequency lies within our test range.