Lie Detection Through Voice Stress Analysis

Jithin N R[1] Nandan M R[2] Eldho George[3] Akhil P Ramesh[4] Sreehari V R[5]
[1,2,3,4]UG Student [5]Assistant Professor
[1,2,3,4,5]Department of Electronics and Communication Engineering
[1,2,3,4,5]Matha College of Technology

Abstract— The proposed paper discusses about the MATLAB approach for detection of deception. The audio of the person is recorded in real time and is converted into wav format. This wav file is then loaded and run using a MATLAB code which detects whether there is deception. The dime-domain features and frequency-domain features are analyzed in order to detect variations in the recorded voice file.

Key words: Deception, Time-Domain Features, Frequency Domain Features

I. INTRODUCTION

Lying is not a rare incident in human life. People tend to lie if they realize that they can gain from doing so. In fact people believe that lying in a necessary situation is not a big offence. Most often people lie during an interview, when asked about their hobbies, etc.

People lie in order to make themselves appear better than others. But this very tendency of humans become a headache during investigation of a crime. In such situation truth detection becomes unavoidable. Over these years many cases were solved with the help of truth detection. From early days onwards polygraph test has been used for deception detection. The polygraph is a set of equipment that accurately measures various sorts of bodily activity such as heart rate, blood pressure, respiration and palmar sweating. This bodily activity can be displayed via ink writing pens on to charts or via a computer's visual display unit.

In this paper a MATLAB approach has been accommodated. The suspect will be asked some questions. The questions regarding the crime, questions like the suspects name, age will be asked. These questions are asked in order to get the variation in their voice while speaking truth and while the person is under stress. In moments of stress, as when you tell a lie and that you dare not to get caught, the body prepares for fight by increasing the readiness of its muscles to spring into action. Thus the muscle vibration increases. All muscles in the body, including the vocal chords vibrate in the 8 to 12 HZ range. When the person lies, the vibration increases than the normal rate and this creates stress in the voice. By evaluating this, it can be determined that the person is telling the truth or not. The time-domain audio features and frequency-domain audio features are extracted from the recorded voice in order to achieve the variation pattern.

The time-domain audio features include energy, entropy of energy, zero-crossing rate. While the frequency-domain audio features include spectral centroid and spread, spectral entropy, spectral flux, spectral roll-off.

II. BLOCK DIAGRAM

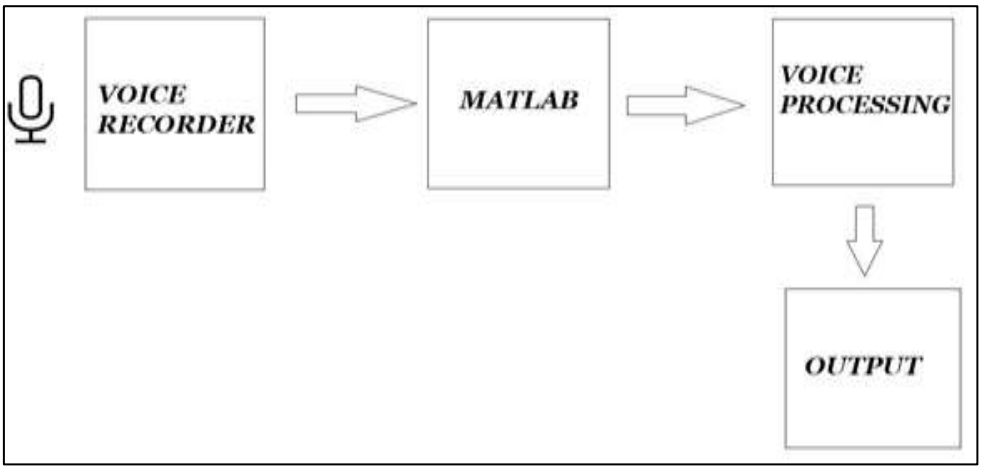

Fig. 1: Block diagram of the proposed method

III. FLOWCHART

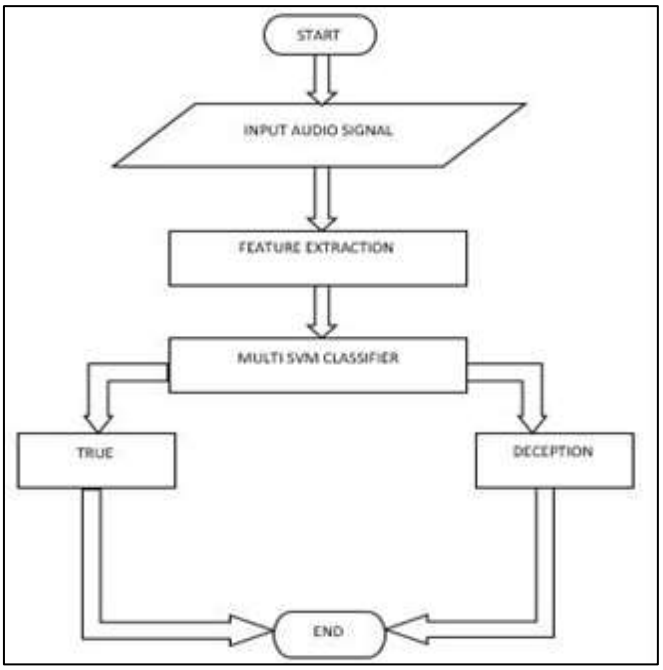

Fig. 2: Flowchart of proposed method

A. Time-Domain Audio Features

Time-domain audio features are extracted directly from the samples of audio signal. What follows are some of the most important time-domain features.

1) Energy

Let xi (n), n=1…, WL be the sequence of audio samples of the *i*th frame, where WL is the length of the frame. The short term energy is computed according to the equation

$$E(i) = \sum_{n=1}^{W_L} |x_i(n)|^2.$$

Usually, energy is normalized by dividing it with WL to remove the dependency on the frame length. Therefore, the above equation becomes,

$$E(i) = \frac{1}{W_L} \sum_{n=1}^{W_L} |x_i(n)|^2.$$

The following function extracts the energy value of a given audio frame:

Function E = feature_energy (window)

function E=feature_energy (window);

%this function calculates the energy of an audio frame.

ARGUMENTS:

- window: an array that contains the audio samples of the input frame.

RETURN:

- E: the computed energy value

%

E=(1/length(window)) * sum(abs(window. ^2));

2) Zero-Crossing Rate

The zero-crossing rate of an audio frame is the rate of sign-changes of the signal during the frame. In other words, it is the number of times the signal changes value, from positive to negative and vice-versa, divided by the length of the frame. The ZCR is defined according to the following equation:

$$Z(i) = \frac{1}{2W_L} \sum_{n=1}^{W_L} | \, sgn[x_i(n)] - sgn[x_i(n-1)] \, |,$$

The computation of the zero-crossing rate for a given frame is implemented in the following m-file:

Function z =feature_zcr (window); % function z= feature_zcr (window);

this function calculates the zero crossing rate of an audio frame.

ARGUMENTS:

- window: an array that contains the audio samples of the input frame.

RETURN:
- z: the computed zero crossing rate value
window2 = zeroes (size(window)); Window2 (2:end) = window(1:end-1);
Z=(1/(2*length(window)))*sum(abs(sign(window)-sign(window2)));

ZCR can be interpreted as a measure of the noisiness of a signal. For example it exhibits higher value in the case of noisy signals.

3) Entropy Of Energy

The short-term entropy of energy can be interpreted as a measure of abrupt changes in the energy level of an audio signal. In order to compute it, we first divide each short-term frame in K sub-frames of fixed duration. Then for each sub-frame, j, we compute its energy and divide it by the total energy of the short-term frame. The division operation is a standard procedure and serves as a means to treat the resulting sequence of sub-frame energy values ej, j= 1…., K as a sequence of probabilities.
H (i) = - ∑ ej log2 (ej)
The following function computes the entropy of energy of a short-term audio frame:
Function entropy = feature_energy_entropy(window, numoshortblocks) %function entropy =feature_energy_entropy(window, numofshortblocks)
%this function computes the energy entropy of the given frame.
%ARGUMENTS
- window: an array that contains the audio samples of the input frame
- numofshortblocks: number of sub-frames
used in entropy computation
%
%RETURN
- entropy : the energy entropy value
%total frame energy:
Eol = sum(window.^2);
Winlength = length(window);
Subwinlength = floor(winlength / numofshortblocks);
If length (window) ≠ subwinlength * numofshortblocks
Window = window(1: subwinlength I numofshortblocks); %end
%get sub-window:
Subwindow = reshape(window, subwinlength, numofshortblocks);
% compute normalized sub-frame energies:
S= sum(subwindow.^2 / (Eol+eps);
% compute entropy of the normalized sub-frame energies: Entropy = -sum(s.*log2(s+eps));

B. Frequency-Domain Audio Features

DFT is widely used in audio signal analysis because it provides a convenient representation of the distribution of the frequency content of the sounds, i.e., of the sound spectrum. Audio features based on the DFT of the audio signal is also known as frequency-domain (or spectral) audio features. Some of the frequency-domain audio features are as follows.

1) Spectral Centroid And Spread

The spectral centroid and spread are two simple measures of spectral position and shape. The spectral centroid is the center of gravity of the spectrum. The value of spectral centroid, Ci, of the *i*th audio frame is defined as:

$$C_i = \frac{\sum_{k=1}^{W_{fL}} kX_i(k)}{\sum_{k=1}^{W_{fL}} X_i(k)}.$$

Spectral spread is the second central moment of the spectrum. In order to compute it, one has to take the deviation of the spectrum from the spectral centroid, according to the following equation:

$$S_i = \sqrt{\frac{\sum_{k=1}^{W_{fL}} (k - C_i)^2 X_i(k)}{\sum_{k=1}^{W_{fL}} X_i(k)}}.$$

The MATLAB code that computes the spectral centroid and spectral spread of an audio frame is as follows.
function C = SpectralCentroid(signal,windowLength, step, fs) signal = signal / max(abs(signal));
curPos = 1;
L = length(signal);
numOfFrames = floor((L-windowLength)/step) + 1; H = hamming(windowLength);
m = ((fs/(2*windowLength))*[1:windowLength])'; C = zeros(numOfFrames,1);
for (i=1:numOfFrames)
window = H.*(signal(curPos:curPos+windowLength-1)); FFT = (abs(fft(window,2*windowLength)));
FFT = FFT(1:windowLength); FFT = FFT / max(FFT);

C(i) = sum(m.*FFT)/sum(FFT); if (sum(window.^2)<0.010)
C(i) = 0.0; end
curPos = curPos + step; end
C = C / (fs/2);

2) Spectral Entropy

Spectral entropy is computed in a similar manner to the entropy of energy, although, this time, the computation takes place in the frequency domain. The entropy of the normalized spectral energy nf is computed according to the equation:

$$H = -\sum_{f=0}^{L-1} n_f \cdot \log_2(n_f).$$

The function that implements the spectral entropy is as follows:

Function En = feature_spectral_entropy(windowFFT, numofshortblocks) %function En = feature_spectral_entropy(windowFFT, numofshortblocks)
%
% this function computes the spectral entropy of the given %audio frame
%
%ARGUMENTS
% - windowFFT: the abs(FFT) of an audio frame
% (computed by getDFT() function)
%- numofshortbins: the number of bins in which the spectrum %is divided
%
%RETURN:
% -En : the value of the spectral entropy
%
%number of DFT coefs Fftlength = length(windowFFT);
% total frame (spectral) energy
Eol = sum(windowFFT.^2); % length of sub-frames:
Subwinlength = floor(fftlength / numofshortblocks);
If length(windowFFT) ≠ subwinlength * numofshortblocks windowFFT = windowFFT(1: subwinlength * numofshortbloks);
end
%define sub-frames:
Subwindow = reshape(windowFFT, subwinlength, numofshortblocks);
% compute spectral sub-energies: S= sum(subwindow.^2) / (Eol+eps);
% compute spectral entropy:
En = -sum(s.*log2(s+eps));

3) Spectral Flux

Spectral flux means the spectral changes between two successive frames and is computed as the squared difference between the normalized magnitudes of the spectra of the two successive short-term windows:

$$Fl_{(i,i-1)} = \sum_{k=1}^{W_{fL}} (EN_i(k) - EN_{i-1}(k))^2.$$

The spectral flux has been implemented in the following function:

function F = SpectralFlux(signal,windowLength, step, fs) signal = signal / max(abs(signal));
curPos = 1;
L = length(signal);
numOfFrames = floor((L-windowLength)/step) + 1; H = hamming(windowLength);
m = [0:windowLength-1]'; F = zeros(numOfFrames,1); for (i=1:numOfFrames)
window = H.*(signal(curPos:curPos+windowLength-1)); FFT = (abs(fft(window,2*windowLength)));
FFT = FFT(1:windowLength); FFT = FFT / max(FFT);
if (i>1)
F(i) = sum((FFT-FFTprev).^2); else
F(i) = 0; end
curPos = curPos + step; FFTprev = FFT;
end

4) Spectral Rolloff

This feature is defined as the frequency below which a certain percentage of the magnitude distribution of the spectrum is concentrated. Therefore, if the mth DFT coefficient corresponds to the spectral roll off of the ith frame, then the following equation is used:

$$\sum_{k=1}^{m} X_i(k) = C \sum_{k=1}^{W_{fL}} X_i(k),$$

Where C is the adopted percentage. The spectral roll off frequency is usually normalized by dividing it with WfL, so that it takes values between 0 and 1. The MATLAB function that implements this feature is as follows:

```
function mC = SpectralRollOff(signal,windowLength, step, c, fs) signal = signal / max(abs(signal));
curPos = 1;
L = length(signal);
numOfFrames = (L-windowLength)/step + 1; H = hamming(windowLength);
m = [0:windowLength-1]'; for (i=1:numOfFrames)
window = (signal(curPos:curPos+windowLength-1)); FFT = (abs(fft(window,512)));
FFT = FFT(1:255); totalEnergy = sum(FFT); curEnergy = 0.0; countFFT = 1;
while ((curEnergy<=c*totalEnergy) && (countFFT<=255)) curEnergy = curEnergy + FFT(countFFT);
countFFT = countFFT + 1; end
mC(i) = ((countFFT-1))/(fs/2); curPos = curPos + step;
end
```

IV. WORKING

In the proposed method the voice of the person is recorded in real time. This voice is then converted into wav file format as the main program only reads the wav file. For the main code to read the recorded wav file, the program must be trained. The training function is done again by another code designated to do it. While training, the feature matrix in which the features are stored will be automatically updated. The person will be asked some questions which include questions of which the answer is known. These questions are asked in order to know the voice pattern while the person is telling the truth. Once the program is trained then the program can be run to check for deception from the recorded voice wav file. The output will be shown in the output window.

V. CONCLUSION

In this paper, we have proposed a reliable method for deception detection. Normally used technique such as polygraph method is time consuming compared to this proposed method. In this MATLAB approach the voice is recorded and then converted to a wav file to run using the MATLAB program. The proposed system works based on the voice stress detection. Time-domain audio features and frequency-domain audio features are analyzed to obtain the voice stress pattern. The proposed method helps in deception detection at a lower cost with a very sufficient accuracy.

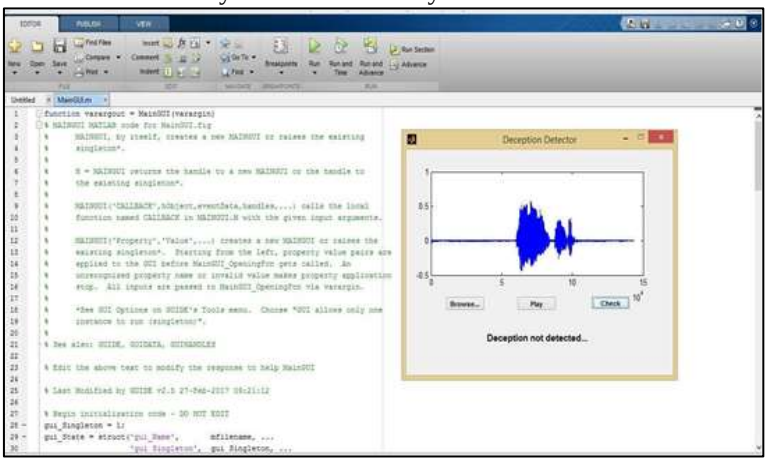

Fig. 3: screen shot of MATLAB when deception is not detected

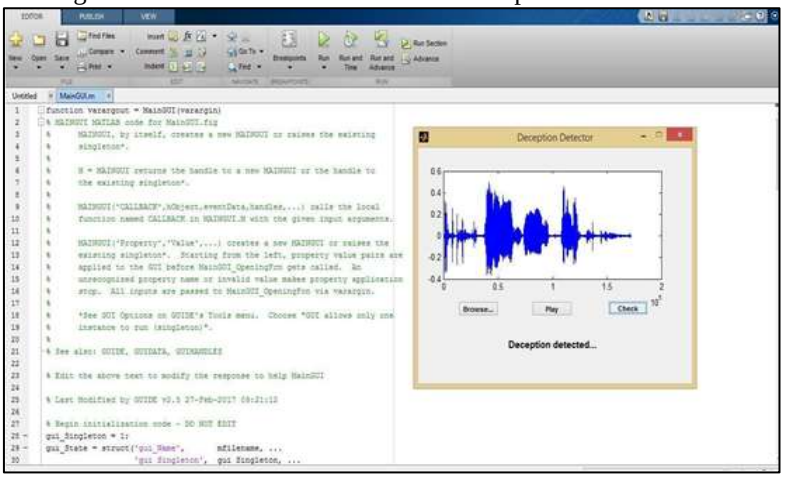

Fig. 4: screen shot of MATLAB when deception is detected

REFERENCES

[1] "Voice stress analysis and evaluation ", Darren M Haddad, Roy Ratley. Air Force Research Laboratory/IFE, 32 Brooks Road, NY 13441.
[2] "The truth and nothin but truth: multimodal analysis for deception detection". Mimansa Jaiswal, institute of engineering and technology DA University, Indore, Rajiv Bajpai,school of computer science and engineering, Nauang Technologial University, Singapore, Sairam Tabibu, Electronics Engineering, IIT BHU, Varanasi, India.
[3] "Voice stressdetection: a method for stress analysisi detecting fluctuations on lippold microtremor spectrum using FFT" Roberto Cabrera Cosetl and J.M. David Baez Lopez. Departmento de Computacion Electronica y Mecatronica Universidad de las Americas puebla, Mexico.
[4] Olof Lippold, "physiological Tremor," Scientific American, volume 224, November 3, March 1971.