Snickometer Edge Detection

EE555 Lab Project

Nikhil N and TimJim K Momin
M.Tech in Systems Control and Automation, IIT Guwahati

20 Jun 2020

1 Introduction

The Snickometer which is commonly known as Snicko is used in televising cricket to graphically analyse sound and video, and show whether a fine noise or snick occurs as ball passes bat. But the technical faults leading to wrong decision making are quite common in cricket game. In this project, advanced audio analysis using scientific python is used to take a more reliable decision. Feature extractions like Short Time Fourier Transform (STFT), Spectral centroid, Spectral rolloff, Tempo and Zero crossing rate are used to precisely distinguish between the ball-bat and ball-pad or other snick.

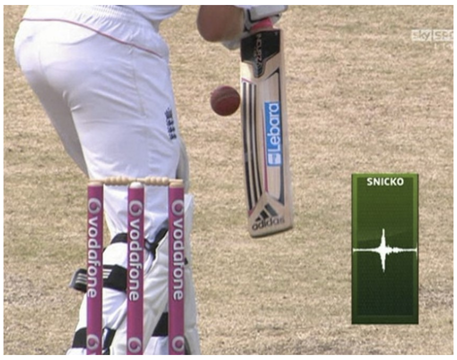


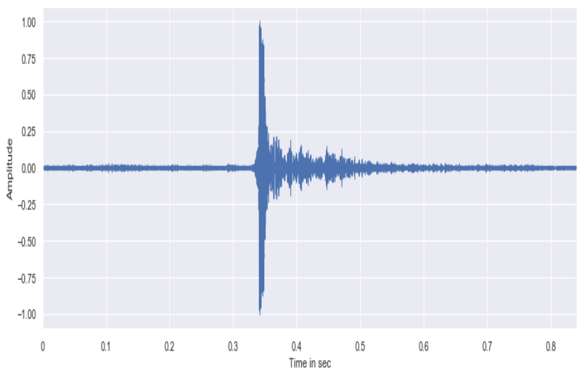
Figure 1: Real time Snicko

2 Theory

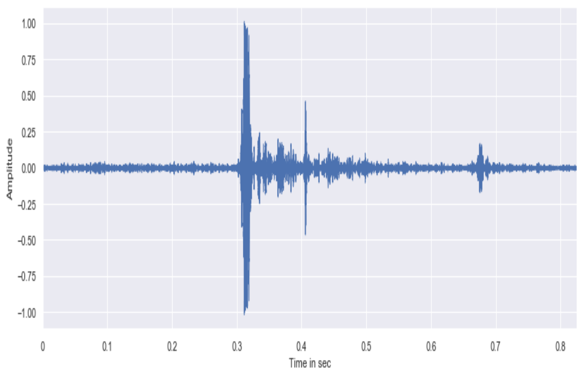
We have considered three signals Ball-bat snick, Ball-pad snick and the Ambient noise to develop the algorithm. Initially the raw signal is visualised in the time domain and noise filtering is done using the spectral gating, an algorithm similar to the one outlined by audacity for noise reduction. Then feature extraction is done on the noise free signal. That is Frequency domain plot, STFT, Spectral centroid, Spectral rolloff, Zero crossing rate and Tempo are used to characterise the Ball-bat and Ball-pad snick, then finally to come up with a the right decision.

2.1 Visualisation

Following are the time domain visualisation of Ball-bat and Ball-pad snicks. Here the signals are raw signals with noise and amplitudes are normalised.



(a) Ball-Bat Snick



(b) Ball-Pad Snick

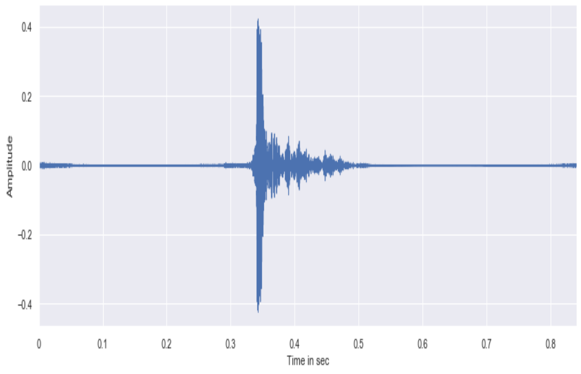
Figure 2: Time Domain Representation

2.2 Removing the Noise

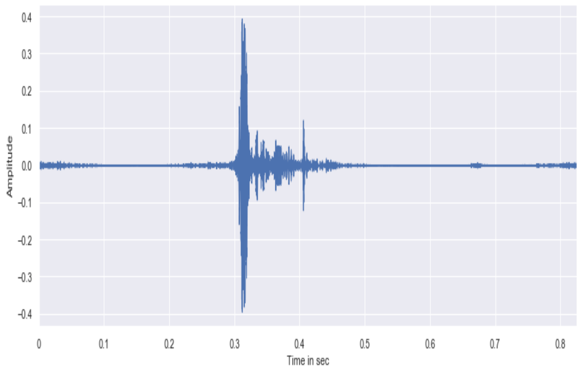
Noise reduction is done using spectral gating, the algorithm for this is based on the one outlined by famous audio editor audacity.

Algorithm

- An FFT is calculated over the noise audio clip
- Statistics are calculated over FFT of the the noise (in frequency)
- A threshold is calculated based upon the statistics of the noise (and the desired sensitivity of the algorithm)
- An FFT is calculated over the signal
- A mask is determined by comparing the signal FFT to the threshold
- The mask is smoothed with a filter over frequency and time
- The mask is appled to the FFT of the signal, and is inverted



(a) Ball-Bat Snick



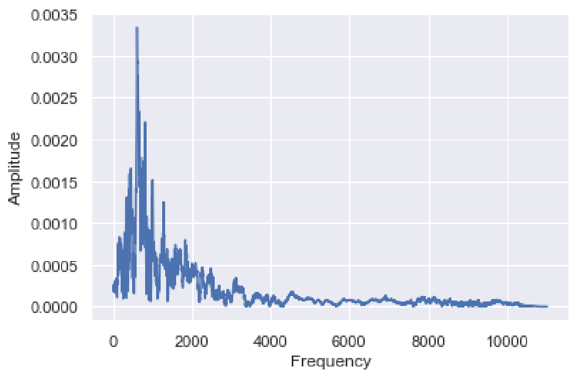
(b) Ball-Pad Snick

Figure 3: Time Domain Spectrum After Noise Removal

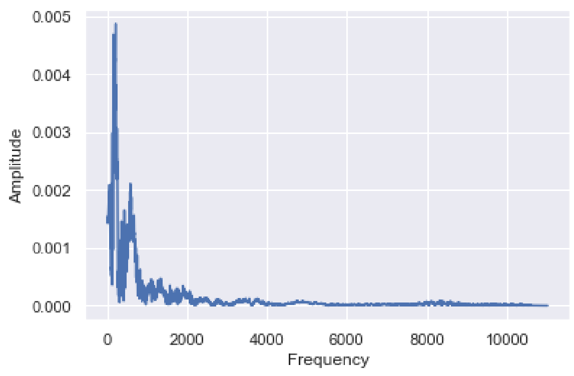
Note: The python implementation of the same is available at https://github.com/timsainb/noisereduce It is utilised here with necessary modifications.

2.3 Frequency Spectrum

Figure below shows the Frequency spectrum for both signals. It is found that the Ball-Bat snick is having wider frequency content than the Ball-Pad snick, which got a little concentrated response.



(a) Ball-Bat Snick



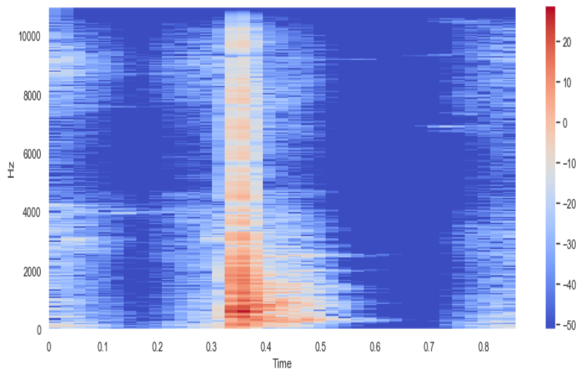
(b) Ball-Pad Snick

Figure 4: Frequency Spectrum

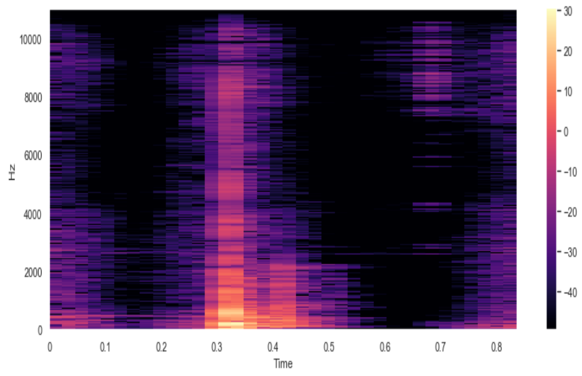
2.4 Feature Extraction

2.4.1 Spectrogram

The Short-time Fourier transform (STFT), is a Fourier-related transform used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time. In practice, the procedure for computing STFTs is to divide a longer time signal into shorter segments of equal length and then compute the Fourier transform separately on each shorter segment. This reveals the Fourier spectrum on each shorter segment. One then usually plots the changing spectra as a function of time, known as a spectrogram. Figure below shows the Spectrogram of both snicks.



(a) Ball-Bat Snick



(b) Ball-Pad Snick

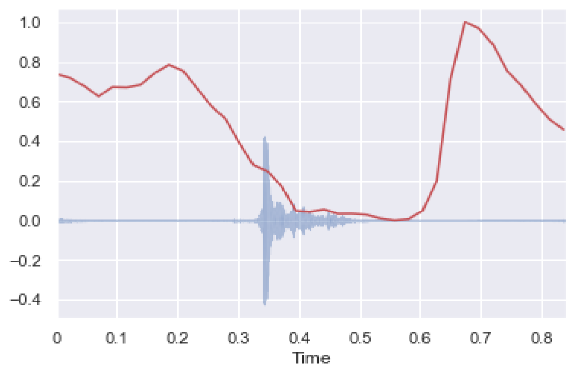
Figure 5: Spectrogram

2.4.2 Spectral Centroid

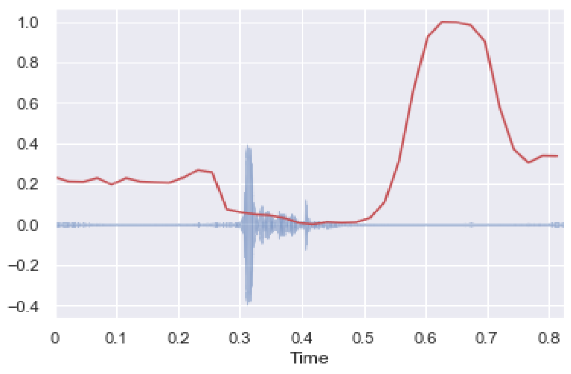
The spectral centroid is a measure used in digital signal processing to characterise a spectrum. It indicates where the center of mass of the spectrum is located. Perceptually, it has a robust connection with the impression of brightness of a sound. It is calculated as the weighted mean of the frequencies present in the signal, determined using a Fourier transform, with their magnitudes as the weights.

$$\text{Centroid} = \frac{\sum_{n=0}^{N-1} f(n)x(n)}{\sum_{n=0}^{N-1} x(n)}$$

where $x(n)$ represents the weighted frequency value or magnitude of bin number n and $f(n)$ represents the center frequency of that bin. Figure below shows the Spectral centroid of snicks.



(a) Ball-Bat Snick

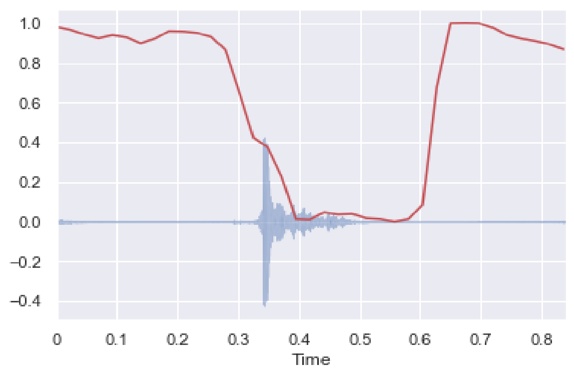


(b) Ball-Pad Snick

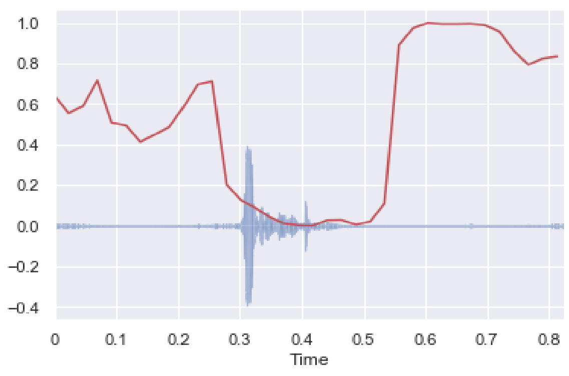
Figure 6: Spectral Centroid

2.4.3 Spectral rolloff

Spectral rolloff is the frequency below which a specified percentage of the total spectral energy, e.g. 85%, lies. Here c rolloff frequency for each frame in the signal is calculated. Figure below shows the Spectrogram of both snicks.



(a) Ball-Bat Snick

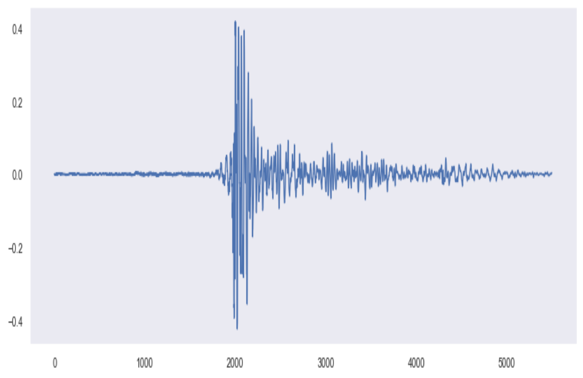


(b) Ball-Pad Snick

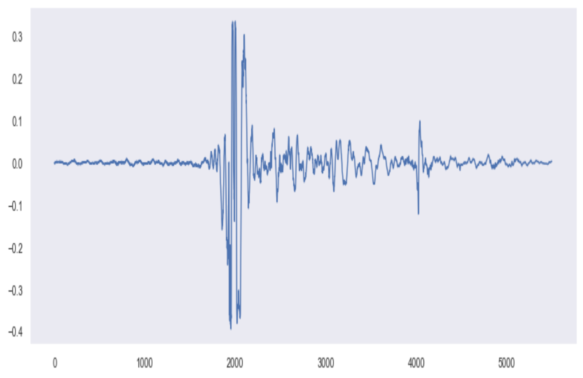
Figure 7: Spectral rolloff

2.4.4 Zero Crossing Rate

Zero-crossing rate is a measure of number of times in a given time interval or frame that the amplitude of the signals passes through a value of zero. Figure below shows the Frames of snicks used for determining zero crossing rate. It is found that the zero crossing rate is more for Ball-Bat snick than the Ball-Pad snick.



(a) Ball-Bat Snick



(b) Ball-Pad Snick

Figure 8: Zero Crossing Rate

2.4.5 Tempo

The tempo of a piece of sound signal is the speed of the underlying beat. Tempo is measured in BPM, or beats per minute

2.5 Numerical Extraction

Some twenty samples, ten from each snick are considered to numerically outline the range in which the above mentioned characteristics falls. It is consolidated in the table below.

Feature	*Spectral Centroid*	*Spectral Rolloff*	*Zero Crossing*	*Tempo*
Bat-Ball	2500-3500	5800-7400	500-1200	100-250
Bat-Pad	2900-3900	5500-7100	200-500	100-230

Table 1: Numerical Range of Features

2.6 Program using python

```python
"""
Created on Sat Jun 13 23:00:50 2020

@author: Nikhil

"""
#%% IMPORT LIBRARIES

import IPython
import sklearn
import scipy
from scipy.io import wavfile
import scipy.signal
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import librosa
import librosa.display
import noisereduce as nr
import os

#%% PREPARING DATA

# Load Data
wav_loc = "wav/ballonpad5.wav"
data, sr = librosa.load(wav_loc)

# Display data
print('Raw Signal')
sns.set()
plt.figure(figsize=(20,4))
librosa.display.waveplot(data, sr=sr)
plt.xlabel("Time in sec")
plt.ylabel("Amplitude")
plt.show()

# Removing Noise
nois_loc = "wav/noise.wav"
ns_data, ns_sr = librosa.load(wav_loc)
noise_clip = ns_data[: len(data)]
snr = 1                              # signal to noise ratio
noise_clip = noise_clip / snr
output = nr.reduce_noise(audio_clip=data, noise_clip=noise_clip,verbose=False)
max_ind= output.argmax()
print('Max value: '+str(max_ind))

# Display noise removed data
print('Noise Removed Signal')
plt.figure(figsize=(20,4))
librosa.display.waveplot(output, sr=sr)
plt.xlabel("Time in sec")
plt.ylabel("Amplitude")
plt.show()

#%% FEATURE EXTRACTION

# Getting dimension of array
print('Dimension: '+ str(data.shape))
print('Sampling Rate: '+ str(sr))

# Frequency domain plot
def fft_plot(audio,samp_rate):
    n=len(audio)
    T=1/samp_rate
    yf=scipy.fft(audio)
    xf=np.linspace(0.0,1.0/(2*T),n/2)
    fig,ax =plt.subplots()
    ax.plot(xf, 2.0/n * np.abs(yf[:n//2]))
    sns.set()
    plt.xlabel("Frequency")
    plt.ylabel("Amplitude")
    return plt.show()
print('Frequency spectrum')
fft_plot(output,sr)
```

```

# Zoom in
if (max_ind-2000 >0):
    n0=max_ind-2000
else:
    n0=0
if (max_ind+3500 < len(output)):
    n1=max_ind+3500
else:
    n1=len(output)
print('Zoomed Signal')
plt.figure(figsize=(14, 5))
plt.plot(output[n0:n1])

# Spectrogram
X = librosa.stft(output)
Xdb = librosa.amplitude_to_db(abs(X))
plt.figure(figsize=(14, 5))
librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
plt.colorbar()
print('Spectrogram')
plt.show()

# Spectral centroid
print('Spectral centroid ')
spectral_centroids = librosa.feature.spectral_centroid(output, sr=sr)[0]
spectral_centroids.shape
# Computing the time variable for visualization
frames = range(len(spectral_centroids))
t = librosa.frames_to_time(frames)
# Normalising the spectral centroid for visualisation
def normalize(output, axis=0):
    return sklearn.preprocessing.minmax_scale(output, axis=axis)
#Plotting the Spectral Centroid along the waveform
librosa.display.waveplot(output, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_centroids), color='r')
plt.show()

# Spectral roll off
print('Spectral roll off')
spectral_rolloff = librosa.feature.spectral_rolloff(output, sr=sr)[0]
librosa.display.waveplot(output, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_rolloff), color='r')
plt.show()

# Displaying  the MFCCs:
print('MFCCs')
mfccs = librosa.feature.mfcc(output, sr=sr)
print(mfccs.shape)
librosa.display.specshow(mfccs, sr=sr, x_axis='time')

#%% EXTRACTING NUMERICAL VALUES

# Spectral centroid
cent = librosa.feature.spectral_centroid(y=output, sr=sr)
cent_mean=np.mean(cent)
cent_std=np.std(cent)
cent_skew=scipy.stats.skew(cent,axis=1)[0]
print('Mean: '+str(cent_mean))
print('SD: '+str(cent_std))
print('Skewness: '+str(cent_skew))

# Spectral roll off
rolloff = librosa.feature.spectral_rolloff(y=output, sr=sr)
rolloff_mean=np.mean(rolloff)
rolloff_std=np.std(rolloff)
rolloff_skew=scipy.stats.skew(rolloff,axis=1)[0]
print('Mean: '+str(rolloff_mean))
print('SD: '+str(rolloff_std))
print('Skewness: '+str(rolloff_skew))

# Zero crossings
zero_crossings = librosa.zero_crossings(output[n0:n1], pad=False)
zero_cross=sum(zero_crossings)
print('Zero crossing: '+str(zero_cross))

```

```python
# Tempo
y_harmonic, y_percussive = librosa.effects.hpss(output)
tempo, beat_frames = librosa.beat.beat_track(y=y_harmonic, sr=sr)
print('Detected Tempo: '+str(tempo)+ ' beats/min')

#%% TAKING DECISION

import os
if(cent_mean>2500 and cent_mean < 4000 and rolloff_mean >5500 and rolloff_mean < 7500 ):
    ball_hit=True
else:
    ball_hit=False

if((tempo >100 and tempo < 250) and (zero_cross < 550 ) and ball_hit ):
    decision="Not Out!!"
    face='\N{smiling face with smiling eyes}'

elif((tempo > 100 and tempo < 300) and (zero_cross > 550) and ball_hit):
    decision="Out!!"
    face='\N{pensive face}'
else :
    decision="oops!!"
    face='\N{upside-down face}'

if os.path.exists("control.txt"):
    os.remove("control.txt")
f = open("control.txt","w")
f.write(decision)
f.close()
print (decision)
print(" "+face)
```

The program primarily utilised the *librosa* library for audio analysis and the final decision is stored in a text file named *control.txt*, which could be read by other programs to have further ations like rotating the servo motor attached to the score board.

2.7 Program to run Servo Motor using Arduino

```cpp

#include <Arduino.h>
#include <Servo.h>

Servo servo;
int pos;

void setup() {
  pinMode(12,INPUT);
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
  servo.attach(9); //attach it to pin 9
}

void loop() {
    if(digitalRead(12)==HIGH ){
    digitalWrite(13, HIGH);                  //ON led
    for (pos =0; pos <= 360; pos += 1) {     // goes from 0 degrees to 360 degrees
      servo.write(pos);                      // MOVE to position 'pos'
      delay(15);                             // wait 15ms for the servo to reach the position
      }
    }
  else{
     digitalWrite(13, LOW);                 //OFF led
    }
    delay(1);
}
```

The program continuously monitor the pin number 12, whenever it goes HIGH arduino will rotate the servo motor by 360 deg, i.e. one complete revolution. So the decision file obtained from the python program, control.txt can be used to give a HIGH or LOW input to the arduino. That is when the decision is 'Out', we will give HIGH input to pin 12, while when the decision is 'Not Out' we will give a LOW input to the pin 12.

3 Results

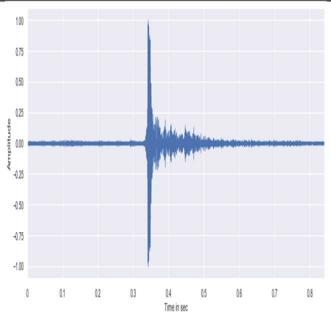
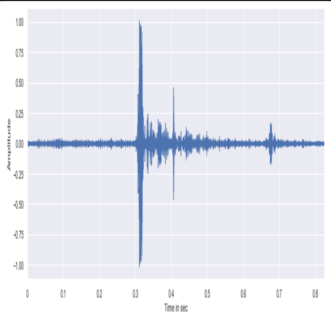
Feature	*Signal*	*Spectral Centroid*	*Spectral Rolloff*	*Zero Crossing*	*Tempo*	*Decision*
Bat-Ball		3238.65	6585.95	747	215.33	Out
Bat-Pad		3878.07	7015.40	400	107.66	Not Out

Table 2: The Test Result