Digital Signal Processing Lab

Demo 7 - Exercise 4 (Microphone AM)

Saad Zubairi
shz2020

September 24, 2025

Solution

In this solution, we modify the `mic_filter.py` program so that instead of filtering the input microphone signal, it applies amplitude modulation at carrier frequency $f_0 = 400$ Hz. The resulting signal is

$$y(t) = x(t)\cos(2\pi f_0 t),$$

which is then played through the loudspeaker and is saved as a wav file.

For this we made the following changes to the original code:

- Removed the difference equation and replaced it with a cosine modulation at $f_0 = 400$ Hz:

```
    # Amplitude modulation: y[n] = x[n] * cos(2*pi*f0*n/RATE)
    modulation = math.cos(2.0*math.pi*f0*n/RATE)
    y0 = x0 * modulation

```

Snippet 1: AM modulation

- Added code to save the output to a wave file:

```
    # Open output wave file
    wf = wave.open('mic_am_400Hz.wav', 'wb')
    wf.setnchannels(CHANNELS)
    wf.setsampwidth(WIDTH)
    wf.setframerate(RATE)

```

Snippet 2: Opening/writing output wav file stream

```
    stream.write(output_bytes)
    wf.writeframes(output_bytes)

```

Snippet 3: Writing bytes in the wav file

- Left the rest of the code (input, stream I/O, clip16, etc.) unchanged.

The generated audio has a metallic or robotic quality. The speech becomes less natural because its spectrum is mirrored around 400 Hz, creating a buzzy timbre.

Full code

```python
import pyaudio
import struct
import math
import wave

def clip16( x ):
    # Clipping for 16 bits
    if x > 32767:
        x = 32767
    elif x < -32768:
        x = -32768
    else:
        x = x
    return (x)

WIDTH       = 2         # Number of bytes per sample
CHANNELS    = 1         # mono
RATE        = 16000     # Sampling rate (frames/second)
DURATION    = 6         # duration of processing (seconds)

N = DURATION * RATE     # N : Number of samples to process

f0 = 400.0              # Hz

p = pyaudio.PyAudio()

# Open audio stream
stream = p.open(
    format      = p.get_format_from_width(WIDTH),
    channels    = CHANNELS,
    rate        = RATE,
    input       = True,
    output      = True)

# Open output wave file
wf = wave.open('mic_400hz_output.wav', 'wb')
wf.setnchannels(CHANNELS)
wf.setsampwidth(WIDTH)
wf.setframerate(RATE)

print('* Start')

for n in range(0,N):

    # Get one frame from audio input (microphone)
    input_bytes = stream.read(1)
    # If you get run-time time input overflow errors, try:
    # input_bytes = stream.read(1, exception_on_overflow = False)

    # Convert binary data to tuple of numbers
    input_tuple = struct.unpack('h', input_bytes)

    # Convert one-element tuple to number
    x0 = input_tuple[0]

    # Amplitude modulation: y[n] = x[n] * cos(2*pi*f0*n/RATE)
```

```
    modulation = math.cos(2.0*math.pi*f0*n/RATE)
    y0 = x0 * modulation

    # Clip and convert to int16
    output_value = int(clip16(y0))
    output_bytes = struct.pack('h', output_value)

    # Play and save
    stream.write(output_bytes)
    wf.writeframes(output_bytes)

print('* Finished')

stream.stop_stream()
stream.close()
p.terminate()
wf.close()
```

Snippet 4: example code