In lesson one you learned how to record and play audio on the Raspberry Pi. In this lesson we will expand on the topic, covering different audio formats and additional software.

It would be a good idea at this point to cover some scientific aspects of sound. Sound is transmitted through air as a series of waves. We can't see the waves, so it is difficult to visualize them in the mind. In contrast, we have all observed wave phenomena on the surface of water. While the actual physical structure of a water wave is different than a sound wave, we can still use the water wave as an example of wave phenomena.

Let's take the example of waves at the shore of the ocean. We can observe that waves move toward the shore. If we wade out into the water, equipped with a long pole, we can make systematic observations of the water level over time. Suppose we wade out until the water is waist-deep and then plant our pole on the bottom, oriented vertically. As the waves pass by, we notice that the water level marked by our pole rises and falls in a rhythmic pattern. If we are farther from shore than the breaking point of the waves, when a wave passes by, we will not be carried with it. In fact, the water is moving primarily up and down and the effect of the wave is to move our body up and down as we float on the surface. In order to "catch" the wave (body surfing), we need to swim along with the wave until it breaks, at which time it will transport us toward the shore. Until the waves break near shore, the primary movement of water is a rhythmic up and down movement.

The famous 15th century Italian, Leonardo Da Vinci, explained water waves this way: “It often happens that the wave flees the place of its creation, while the water does not; like the waves made in a field of grain by the wind, where we see the waves running across the field while the grain remains in place.” If you grew up on a farm in Kansas, then you would be familiar with this phenomena. If not, this YouTube video might help: https://youtu.be/vYpmMOn3qP0 . Now you can understand the reference to "amber waves of grain" in the song *America the Beautiful*.

In the case of sound, air molecules are transported ***very short*** distances back and forth, perpendicular to the source of the sound. As the air molecules move, zones of increased and reduced density develop, as illustrated in an exaggerated fashion in Figure 1. An animation is available at: http://lafavre.us/sound-wave.gif . It is an illusion that the black dots, representing air molecules, are moving across the page. They are actually moving right then left, short distances (I know because I made the animated graphic).

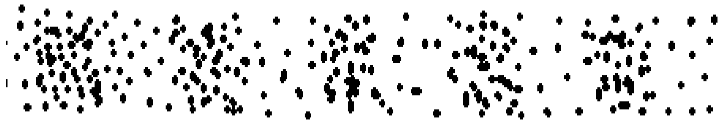

Figure 1 Sound Waves

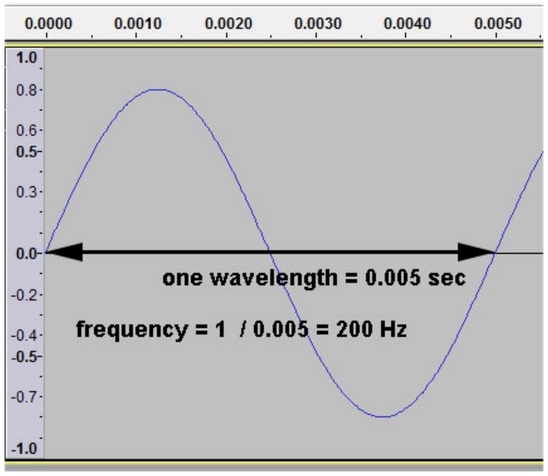

Figure 2 a sine wave

Whether we are talking about water waves or sound waves (or even light), we can represent the waves graphically as in Figure 2. This graphic was generated using the audio recording software named Audacity. I used it to generate a pure tone at a frequency of 200 Hz (200 cycles per second). The graphic includes a measure of the wavelength of the sound (normally we think of wavelength in terms of a distance measurement, but here we are making a measurement in terms of time on the horizontal axis). If one wave passes an observation point in a period of 0.0050 seconds, then 200 waves will pass in a period of one second.

The blue line depicts the shape of the wave. If we are talking about a sound wave, when the wave intersects the vertical scale at a value of 0.0, that would be equivalent to the average density of air (the current barometric pressure). The top of the wave crests at a value of 0.8, which is a measure of wave amplitude. The higher the amplitude, the louder the sound. At the top of the wave, the air is at its greatest density. Then the wave falls and goes below the 0.0 point. When the amplitude has a negative value, then the air has a density less than the average density. If we had a very fancy barometer, that measures air pressure very accurately, we would notice that the air pressure goes up and down rhythmically as sound waves pass by the instrument.

Sound and water waves are analog phenomena. Computers are digital machines and therefore, we need to change the analog sound into a digital record. This is done by sampling the wave pattern. Let's return to the 200 Hz tone. Suppose we sample the sound 2,000 times per second. This is depicted in Figure 3 by applying black dots on the wave form at 0.0005-second intervals (2,000 times per second). At each sample point, the amplitude of the wave is measured and recorded in the digital sound file. If our sampling is adequate, the pattern of the sample dots should closely outline the shape of the wave and we should have a good audio recording.

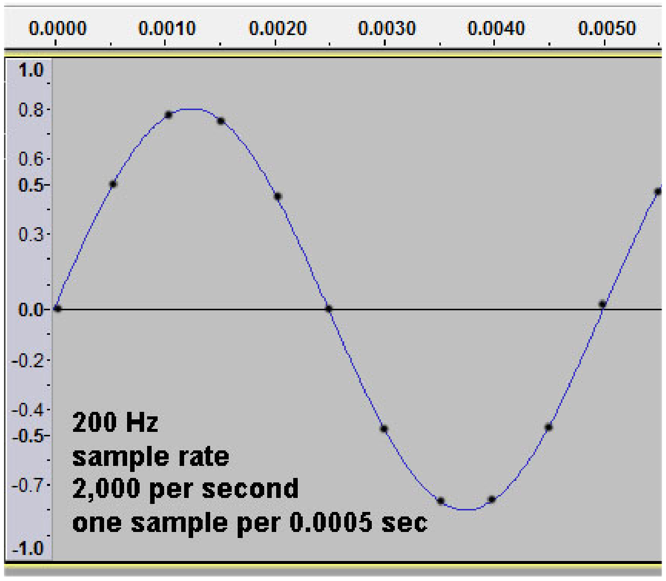

Figure 3 wave sampled 2,000 times per second

Another factor affecting the quality of the sound sampling will be the accuracy of the amplitude measurements. The amplitude is typically measured with 8, 16, 24 or 32 bit values. The higher the bits, the more accurate the amplitude measurement. For 8 bits, the amplitude can have one of 256 values (0 - 255), for 16 bits, 65,536 values are available, etc.

For uncompressed audio formats like wave (.wav), we can estimate the file size of the audio if we know the sampling rate, the bits per sample and the length of the recording in seconds. Suppose we record in wave format at 16,000 samples per second at 16 bits per sample for 10 seconds. The file size will be about 16,000 X 16 X 10 = 2,560,000 bits or 320,000 bytes (about 1/3 MB).

Most sounds we want to record are more complex than a pure tone. Figure 4 is the wave pattern resulting by mixing 200 and 400 Hz tones. Looking at the sampling pattern of 2,000 per second, it seems that perhaps that sampling rate might be enough to reproduce the wave pattern with good fidelity. But most sounds are much more complex than this two-tone sound.

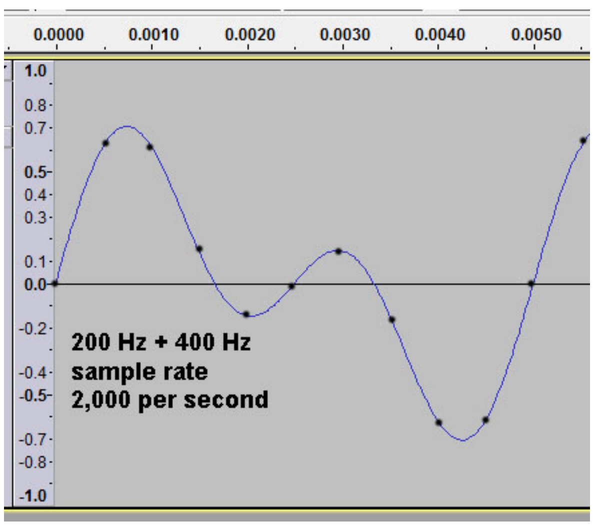

Figure 4 mixed 200 and 400 H tones

Figure 5 is the wave trace of some human speech. Again, I have applied dots on the wave pattern at a rate of 2,000 per second. It is obvious that the speech wave pattern is more complex than the pure tones. It should also be obvious that the sampling rate is not enough to accurately trace out the wave pattern. If we actually recorded human speech at 2,000 samples per second, it is likely we would not be able to understand the words spoken. A sampling rate of 8,000 samples per second is usually considered the minimum required for voice recordings. For most music, the sample rate must be higher, at least 16,000 or 24,000.

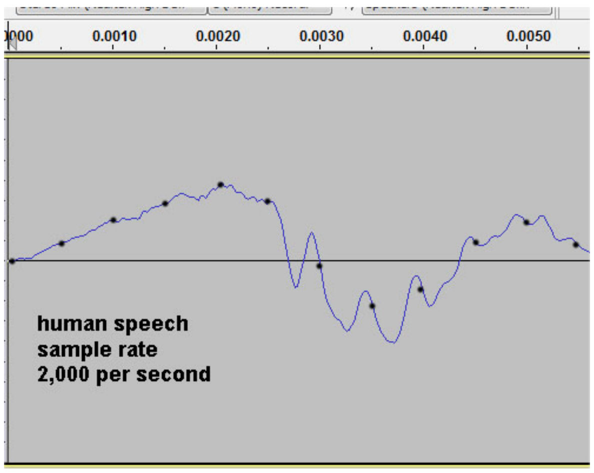

Figure 5 human speech

In the last lesson you learned how to record using the wave audio format. Suppose you want to record some audio and transmit it over the Internet. In order to do this, you want the data rate to be relatively small. For uncompressed wave format, one second of audio at 8,000 sampling rate, 16 bit, would be 8,000 x 16 = 128,000 bits per second (128 kbps). That is a high bit rate just to transport audio. We would be better off using a compressed audio format, like MP3. In this lesson you will discover that MP3 format results is much smaller file sizes than wave. If we want to transport sound quickly, we should be using compressed formats like MP3.

Unfortunately, the ALSA **arecord** method cannot be used to record in MP3 format. However, it is possible to convert the wave audio to MP3 after you record it. To do this we will use the **Lame encoder**.

Install Lame encoder on your Raspberry Pi.

```
sudo apt-get install lame
```

Once you have Lame installed it is very easy to convert wave files to mp3 and it is accomplished quickly. Suppose you have used **arecord** to create a wave file named test.wav. To convert to MP3 enter this in the terminal:

```
lame test.wav test.mp3
```

The above will create a MP3 format audio file by reading test.wav. The wave file will also be retained. Compare the file sizes of the wave and converted MP3 file (open the File Manager, right-click on the audio file, select Properties – the size of file will be listed). I made a 10.6-second wave file, 44,100 sample rate, 16 bit, which resulted in file size of 915 KB. Lame took that file and created a MP3 file of 83.5 KB size. The wave file is about 11 times the size of the MP3 file! But wait, we can do even better.

With **arecord**, we are stuck with a sample rate of 44,100 at 16 bits per sample, which is way overkill for recording voice (a bitrate of 44,100 x 16 = 705,600 bps). Happily, the Lame encoder allows you to specify a bitrate when you use it. You can determine the options available with Lame by issuing this command in the terminal:

```
lame –-help
```

```
pi@GEAR1:~ $ lame --help

LAME 32bits version 3.99.5 (http://lame.sf.net)
usage: lame [options] <infile> [outfile]
    <infile> and/or <outfile> can be "-", which means stdin/stdout.
RECOMMENDED:
    lame -V2 input.wav output.mp3
OPTIONS:
    -b bitrate      set the bitrate, default 128 kbps
    -h              higher quality, but a little slower.  Recommended.
    -f              fast mode (lower quality)
    -V n            quality setting for VBR.  default n=4
                    0=high quality,bigger files. 9=smaller files
    --preset type   type must be "medium", "standard", "extreme", "insane",
                    or a value for an average desired bitrate and depending
                    on the value specified, appropriate quality settings will
                    be used.
                    "--preset help" gives more info on these
    --help id3      ID3 tagging related options
    --longhelp      full list of options
    --license       print License information
```

Of the above options, I have only tried the bitrate. Here is an example:

```
lame –b 8 test.wav test.mp3
```

This creates a MP3 file at bitrate of 8 kbps. I could clearly hear some artifacts in the voice recording but the words are easy to understand. File size is 12 KB. At a bitrate of 16 kbps the quality was better, very little artifact, file size 24 KB (wave file 38 times larger!).

Now that we have an audio file in MP3 format, how do we play it? Unfortunately, the ALSA **aplay** method does not work with MP3 files. However, there is software named **omxplayer** on the Raspberry Pi that will play MP3 files. Suppose we had a file named test.mp3. It would play with following command in the terminal:

```
omxplayer –o local test.mp3
```

The option **–o local** is required to get the sound to playback through the audio/video jack of the Raspberry Pi. Unfortunately, omxplayer does not support USB sound devices. You can see all the options for oxmplayer by using the help option. The result is provided below.

```
pi@GEAR1:~ $ omxplayer -h

  -h  --help                  Print this help
  -v  --version               Print version info
  -k  --keys                  Print key bindings
  -n  --aidx  index           Audio stream index    : e.g. 1
  -o  --adev  device          Audio out device      : e.g. hdmi/local/both/alsa[:device]
  -i  --info                  Dump stream format and exit
  -I  --with-info             dump stream format before playback
  -s  --stats                 Pts and buffer stats
  -p  --passthrough           Audio passthrough
  -d  --deinterlace           Force deinterlacing
      --nodeinterlace         Force no deinterlacing
      --nativedeinterlace     let display handle interlace
      --anaglyph type         convert 3d to anaglyph
      --advanced[=0]          Enable/disable advanced deinterlace for HD videos (default enabled)
  -w  --hw                    Hw audio decoding
  -3  --3d mode               Switch tv into 3d mode (e.g. SBS/TB)
  -M  --allow-mvc             Allow decoding of both views of MVC stereo stream
  -y  --hdmiclocksync         Display refresh rate to match video (default)
  -z  --nohdmiclocksync       Do not adjust display refresh rate to match video
  -t  --sid index             Show subtitle with index
  -r  --refresh               Adjust framerate/resolution to video
  -g  --genlog                Generate log file
  -l  --pos n                 Start position (hh:mm:ss)
  -b  --blank[=0xAARRGGBB]    Set the video background color to black (or optional ARGB value)
      --loop                  Loop file. Ignored if file not seekable
      --no-boost-on-downmix   Don't boost volume when downmixing
      --vol n                 set initial volume in millibels (default 0)
      --amp n                 set initial amplification in millibels (default 0)
      --no-osd                Do not display status information on screen
      --no-keys               Disable keyboard input (prevents hangs for certain TTYs)
```

```
        --subtitles path       External subtitles in UTF-8 srt format
        --font path            Default: /usr/share/fonts/truetype/freefont/FreeSans.ttf
        --italic-font path     Default: /usr/share/fonts/truetype/freefont/FreeSansOblique.ttf
        --font-size size       Font size in 1/1000 screen height (default: 55)
        --align left/center    Subtitle alignment (default: left)
        --no-ghost-box         No semitransparent boxes behind subtitles
        --lines n              Number of lines in the subtitle buffer (default: 3)
        --win 'x1 y1 x2 y2'    Set position of video window
        --win x1,y1,x2,y2      Set position of video window
        --crop 'x1 y1 x2 y2'   Set crop area for input video
        --crop x1,y1,x2,y2     Set crop area for input video
        --aspect-mode type     Letterbox, fill, stretch. Default: stretch if win is specified, letterbox otherwise
        --audio_fifo  n        Size of audio output fifo in seconds
        --video_fifo  n        Size of video output fifo in MB
        --audio_queue n        Size of audio input queue in MB
        --video_queue n        Size of video input queue in MB
        --threshold   n        Amount of buffered data required to finish buffering [s]
        --timeout     n        Timeout for stalled file/network operations (default 10s)
        --orientation n        Set orientation of video (0, 90, 180 or 270)
        --fps n                Set fps of video where timestamps are not present
        --live                 Set for live tv or vod type stream
        --layout               Set output speaker layout (e.g. 5.1)
        --dbus_name name       default: org.mpris.MediaPlayer2.omxplayer
        --key-config <file>    Uses key bindings in <file> instead of the default
        --alpha                Set video transparency (0..255)
        --layer n              Set video render layer number (higher numbers are on top)
        --display n            Set display to output to
        --cookie 'cookie'      Send specified cookie as part of HTTP requests
        --user-agent 'ua'      Send specified User-Agent as part of HTTP requests
        --lavfdopts 'opts'     Options passed to libavformat, e.g. 'probesize:250000,...'
        --avdict 'opts'        Options passed to demuxer, e.g., 'rtsp_transport:tcp,...'
```

For example:

```
    ./omxplayer -p -o hdmi test.mkv
```

Audacity sound recording software

Another option for playback of MP3 files is Audacity. Audacity is a nice sound recorder that will record and playback audio files of various formats. I would recommend that you install it:

```
sudo apt-get install audacity
```

With Audacity you can play MP3 files using USB audio devices.

Creating a Python program for audio recording and playback

There is a very useful method named **call** in the Python module named **subprocess** which we can use to run commands, just as they are entered in the Terminal window. We have used this previously in the lessons covering the temperature sensor. Now you will see how the call method can be used to create Python programs that handle audio recording , playback and format conversion. The first program will start the **arecord** audio recorder and record for 15 seconds, then use **aplay** to playback the recorded audio to USB speakers.

```
#Python 2.7
#record and playback audio
from subprocess import call

print "recorder started"
#duration of 15 is  15 seconds
record = "arecord  --device=hw:1  --format S16_LE  --rate 44100  --duration=15  -c1  python.wav"
call([record], shell=True)
print "recorder stopped -- recording will now play"
playback = "aplay -D sysdefault:CARD=Device_1 python.wav"
call([playback], shell=True)
print "playback finished"
```

This second program records 15 seconds in wave format then converts a copy to MP3 format at a bitrate of 16 kbps.

```
#Python 2.7
#record and convert audio to MP3
from subprocess import call

print "recorder started"
record = "arecord  --device=hw:1  --format S16_LE  --rate 44100  --duration=15  -c1  python.wav"
call([record], shell=True)
print "recorder stopped "
convert = "lame -b 16 python.wav python.mp3"
call([convert], shell=True)
print "audio converted to MP3"
```

You could add to the above program by using omxplayer to playback the converted MP3 audio file, but then you could not use the USB speaker for playback. It may be possible to use command line commands to control the playback with Audacity, but currently this is not fully supported, only experimental (https://manual.audacityteam.org/man/scripting.html).

Using Raspberry Pi Web Browser to play audio

Another way to play audio in wave or mp3 format is with the Chromium web browser. Suppose you had a file named test.mp3 stored in the pi directory. You could play that by entering the path to the file in the web browser address slot like this:

```
file:///home/pi/test.mp3
```

Notice that there are three forward slashes required (///).

Unlike the Omxplayer, the Chromium media player can playback to USB sound devices. However, you must set playback to the USB device BEFORE you open the web browser. In order to do that, right-click on the speaker icon near the upper-right corner of the desktop. There will probably be a green check mark next to "Analog," meaning that playback will be to the audio/video jack on the RPi. If you have a GEAR USB speaker attached to the RPi, it should be listed in the menu as **USB2.0 Device**. Click on that to place the green check mark there. Then open the web browser and enter the address to the local audio file. It should playback to the USB speaker.

Python program to play audio with web browser

Suppose you have developed a Python program to record audio and then automatically upload it to a web server. If you have knowledge of the file name of the audio, then you can write another Python program to play that audio from the server using Chromium.

There is a module in Python named **webbrowser**. The Python program below will open Chromium and play the audio file.

```
import webbrowser

webbrowser.open('put address to audio file here')
```

For example, let us say that a file named test.mp3 is stored in the root directory of this web site: http://my.domain.com Then the code below will play that audio in Chromium on your RPi:

```
import webbrowser

webbrowser.open('http://my.domain.com/test.mp3')
```