The Modal Shop Python Example For Digital Accelerometers And Digital Signal Conditions

Version 6/21/2022

Purpose:
The Modal Shop (TMS) has developed a number of products that utilize the USB Audio Class for data communication. These device interface just like USB Microphones but return other signals than the default acoustic with microphones. The Model 333D01 is a 20g input range accelerometer that returns acceleration samples instead of audio. The 485B39 powers ICP sensors such as microphones and accelerometers and returns voltage. The engineering units can be calculated if the sensor sensitivity is used to convert from volts.

Production specification and other information can be found at: https://www.digiducer.com/

General software development information can be found here:
https://www.digiducer.com/products/software/developers

This will document using Python to acquire, scale, and some basic processing of data from TMS digital audio products.

This example has been developed using the sounddevice package. Sounddevice is based on the PortAudio library. TMS has used PortAudio in other applications and it works well. Other Python audio packages may be usable as well but should be verified to not manipulate the audio data inappropriately.

It is assumed the reader has a basic understand of installing and developing Python. There are many on-line references on Python operation. This guide will focus on the aspects specific to data from the TMS products.

Development Environment:

A. Microsoft Windows 10
B. Python 3.10.5 - https://www.python.org/
C. Sounddevice package version 0.4.4 - https://python-sounddevice.readthedocs.io/en/0.4.4/
D. Numpy package version 1.22.4
E. Matplotlib package version 3.5.1

Reference:

This link is to the reference for sounddevice. It has a number of good examples of general sound access and the definition of the functions. This example was inspired by some of the examples.

https://python-sounddevice.readthedocs.io/en/0.4.4/

General:

The necessary modules are imported at the start of the script. They are expected to already be installed.

Some items the user may want to customize are defined:

```
eu_sen = np.array([100.0, 100.0])
```

```
eu_units = ["g", "g"]
blocksize = 1024 # Number of samples to acquire per block
samplerate = 48000 # 48000, 44100, 32000, 22100, 16000, 11050, 8000
```

eu_sen is engineering units sensitivity in mV. So for a common 100mV/g accelerometer, this is 100.0.

eu_units are the engineering units text units.

blocksize is the number of samples to accumulate and process as a block.

samplerate is the rate to sample data. These are the only rates supported by the hardware. While some Windows Audio API's will allow arbitrary sample rates, use caution as windows will resample the data in ways that don't follow proper signal processing such as anti-alias filtering.

The core of this example is the function TMSFindDevices. It searches for TMS devices and parses the information encoded in the device name. The rest is just standard audio processing. This is unique to our devices. Here is the header from the script.

```
# TMSFindDevices
#
# returns an array of TMS compatible devices with associated
information
#
# Returns dictionary items per device
#   "device"        - Device number to be used by SoundDevice stream
#   "model"         - Model number
#   "serial_number" - Serial number
#   "date"          - Calibration date
#   "format"        - format of data from device, 0 - acceleration, 1
- voltage
#   "sensitivity_int - Raw sensitivity as integer counts/EU ie Volta
or m/s^2
#   "scale"         - sensitiivty scaled to float for use with a
#                     -1.0 to 1.0 scaled data.  Format returned with
#                     'float32' format to SoundDevice stream.
```

The script calls the TMSFindDevices function.

As a simple example, it just uses the first compatible device found. Your specific application can add and option to select a device if multiple devices are attached. Be aware that multiple devices do work but there is no consistent phase between them.

There is some logic to include sensor sensitivity if a digital signal conditioner devices is selected.

A queue is used to return data from the sounddevice callback and it is created.

Some setup for example plots.

The input audio stream is created with sd.InputStream. The device number is determined by the TMSFindDevices function. All devices are 2 channel. Data type 'float32' is generally the most convenient for later processing. The previously defined samplerate and blocksize are used. Sounddevice will accumulate blocksize samples before calling the callback each time. This simplifies having to accumulate variable amounts of data per call which is the default.

'stream.start()' starts the audio devices stream.

The example just loops for 200 iterations. Do whatever is appropriate for your application.

'data = g.get()' populates data with the block of samples received from the call back.

'sdata = data * scale' does a matrix multiply that scales each column by the values in scale. Note: originally it was data *= scale which worked as a calculation. But the plot must use a reference because sometimes it was plotting unscaled data. By using a new variable, that corrected the issue.

The rest is plotting related.

'stream.stop()' stops the stream.