Refactored Analysis Data Capture Code Example

```
from datetime import datetime
import json, math, minimalmodbus
import numpy as np
#import os
#import plotly.graph_objects as go
from scipy.fftpack import fft
from scipy.integrate import cumtrapz
import serial.tools.list_ports
import time
import pandas as pd
from datetime import datetime

def slave_setup(port_str, slave_add):
    # port_str = 'COM#'
    # slave_add: last 2 digits of S/N
    trivibe=minimalmodbus.Instrument(port=port_str, slaveaddress=slave_add)
    # update current slave settings for Tri-Vibe defaults and some useful variables
    trivibe.serial.port                         # this is the serial port name
    trivibe.address                             # this is the slave address (set this to the
last 2 digits of the serial number of the Tri-Vibe that you want to communicate with)
    trivibe.serial.baudrate = 115200            # Baudrate fixed 115200
    trivibe.serial.bytesize = 8                 # Bytesize fixed 8
    trivibe.serial.parity   = "N"               # Parity fixed None
    trivibe.serial.stopbits = 1                 # Stopbits fixed 1
    trivibe.serial.timeout  = 0.10              # Seconds
    trivibe.close_port_after_each_call = True  # Helps communication for Windows Devices (can
be set to false on many Linux devices)
    trivibe.mode = minimalmodbus.MODE_RTU       # modbus mode fixed RTU Mode
    trivibe.clear_buffers_before_each_transaction = True
    return trivibe

def analysis_setup(accelerometer, capture_time_ms, samples_per_axis):
```

```
    trivibe.write_register(32, accelerometer) #DDC_AXIS, 5 == HIGH FREQ
    trivibe.write_register(35, capture_time_ms) #DDC_CAPTURE_TIME_MS
    trivibe.write_long(36, samples_per_axis, signed=False, byteorder=0)
#DDC_HIGH_SAMPLES_PER_AXIS
    trivibe.write_register(33, 1) # DDC_CONTROL_RAW, 1 == MODBUS_CAPTURE_START
    return trivibe

def capture_engine_setup(trivibe):
    t1 = time.time()
    capture_engine_status = trivibe.read_register(34, functioncode=3)
    print('capture_engine_status: '+str(capture_engine_status))
    while capture_engine_status==2:
        capture_engine_status = trivibe.read_register(34, functioncode=3)
        #print('capture_engine_status: '+str(capture_engine_status))
    # show capture engine is complete (capturing done)
    capture_engine_status = trivibe.read_register(34, functioncode=3)
    print('capture_engine_status: '+str(capture_engine_status))
    t2 = time.time()
    #print(t2 - t1)
    return trivibe

def data_collection_raw(trivibe, samples_per_axis):
    # Calculate how many samples are stored on the TriVibe's buffer.
    total_number_of_samples = samples_per_axis*3
    data_array = []
    #print('read start: ' + datetime.now().strftime("%m/%d/%Y_%H:%M:%S"))
    t1 = time.time()
    while len(data_array)<total_number_of_samples:
        read_set = trivibe.read_registers(49, 123, functioncode=3)
        print(read_set[1])
        # remove register 49 from read_set
        read_set.pop(0)
        # take all values from the read_set and add them to our data array
        data_array.extend(read_set)
    # remove any trailing zeroes from the last read_set
    data_array = data_array[0:total_number_of_samples]
    #print('read end: ' + datetime.now().strftime("%m/%d/%Y_%H:%M:%S"))
    t2 = time.time()
    print(t2-t1)
    return data_array
```

```
def split_data_raw(data_array):
    axis_1_data_array_RAW = data_array[0:samples_per_axis] # split data into 3 axes arrays
    axis_2_data_array_RAW = data_array[samples_per_axis:samples_per_axis*2]
    axis_3_data_array_RAW = data_array[samples_per_axis*2:samples_per_axis*3]
    return axis_1_data_array_RAW, axis_2_data_array_RAW, axis_3_data_array_RAW

def get_accelerometer_sensitivity(trivibe):
    s1_a1_mv_per_g = trivibe.read_float(299, functioncode=3, number_of_registers=2,
byteorder=0)
    s1_a2_mv_per_g = trivibe.read_float(301, functioncode=3, number_of_registers=2,
byteorder=0)
    s1_a3_mv_per_g = trivibe.read_float(303, functioncode=3, number_of_registers=2,
byteorder=0)
    s2_a1_mv_per_g = trivibe.read_float(305, functioncode=3, number_of_registers=2,
byteorder=0)
    s2_a2_mv_per_g = trivibe.read_float(307, functioncode=3, number_of_registers=2,
byteorder=0)
    s2_a3_mv_per_g = trivibe.read_float(309, functioncode=3, number_of_registers=2,
byteorder=0)
    high_frequency = [s1_a1_mv_per_g, s1_a2_mv_per_g, s1_a3_mv_per_g]
    low_frequency = [s2_a1_mv_per_g, s2_a2_mv_per_g, s2_a3_mv_per_g]
    return high_frequency, low_frequency

def process_raw_axis(raw_acceleration, sensitivity):
    constant_k = 3000/65535
    virtual_center = (max(raw_acceleration)-min(raw_acceleration))/2+min(raw_acceleration)
    for position in range(len(raw_acceleration)):
        if raw_acceleration[position]>virtual_center:
            raw_acceleration[position] = (abs(raw_acceleration[position] -
virtual_center)*constant_k)/sensitivity # if the value was above virtual center it will be
positive for the timewave form
        elif raw_acceleration[position]<virtual_center:
            raw_acceleration[position] = (-abs(raw_acceleration[position] -
virtual_center)*constant_k)/sensitivity # if the value was below virtual center it will be
negative for the timewave form
        else:
            raw_acceleration[position] = 0 #if the value equals the virtual_center line we
set it to zero for the timewave form
    return(raw_acceleration)

def transform_twf_to_spectrum(twf, capture_time_ms):
```

```
""" Converts Acceleration data to Frequency data and then Charts the Frequency data """
    Fs = samples_per_axis/(capture_time_ms/1000)          # sampling frequency hertz (samples
per second)
    t = np.arange(0,capture_time_ms/1000,1/Fs)            # create a range of numbers from 0
to capture time (in seconds) at a specified interval (1/FrequencySampleRate)
    y = twf                                          # TWF signal
    # convert twf to spectrum
    n = np.size(t)
    Fbin = (Fs/2)*np.linspace(0,1,n//2)
    Y = fft(twf)
    Y_amplitude = (2/n)*abs(Y[0:np.size(Fbin)])
    return Fbin, Y_amplitude

if __name__ == "__main__":
    # Port finder
    ports = list(serial.tools.list_ports.comports())
    accelerometer = 5
    samples_per_axis = 4096

    capture_time_ms = 250
    while len(ports)==0:
        print('Please connect a USB to RS485 serial converter into PC.')
        ports = list(serial.tools.list_ports.comports())
        time.sleep(1)
    else:
        for p in ports:
            print (p)
            continue

    # sensor setup
    trivibe = slave_setup(port_str = 'COM4', slave_add = 99)
    trivibe = analysis_setup(accelerometer = accelerometer, capture_time_ms =
capture_time_ms, samples_per_axis = samples_per_axis)
    trivibe = capture_engine_setup(trivibe)
    #for i in range(5):
    data_array = data_collection_raw(trivibe = trivibe, samples_per_axis =
samples_per_axis)
    axis_1_data_array_RAW, axis_2_data_array_RAW, axis_3_data_array_RAW =
split_data_raw(data_array)

    # timewave form
```

```
    high_frequency, low_frequency = get_accelerometer_sensitivity(trivibe)
    if(accelerometer==5):
        axis_1_twf = process_raw_axis(axis_1_data_array_RAW, high_frequency[0])
        axis_2_twf = process_raw_axis(axis_2_data_array_RAW, high_frequency[1])
        axis_3_twf = process_raw_axis(axis_3_data_array_RAW, high_frequency[2])
    elif(accelerometer==6):
        axis_1_twf = process_raw_axis(axis_1_data_array_RAW, low_frequency[0])
        axis_2_twf = process_raw_axis(axis_2_data_array_RAW, low_frequency[1])
        axis_3_twf = process_raw_axis(axis_3_data_array_RAW, low_frequency[2])
    #print('timewave end:' + datetime.now().strftime("%m/%d/%Y_%H:%M:%S"))

    # spectrum
    axis_1_spectrum = transform_twf_to_spectrum(axis_1_twf, capture_time_ms)
    axis_2_spectrum = transform_twf_to_spectrum(axis_2_twf, capture_time_ms)
    axis_3_spectrum = transform_twf_to_spectrum(axis_3_twf, capture_time_ms)

    # save data to csv
    #df = pd.DataFrame(columns = ['Datetime', 'x_time', 'y_time', 'z_time', 'x_freq',
'y_freq', 'z_freq'])
    df1 = pd.DataFrame({'x_time': axis_1_twf})
    df1['y_time'] = axis_2_twf
    df1['z_time'] = axis_3_twf
    df2 = pd.DataFrame({'x_freq': axis_1_spectrum[1]})
    df2['y_freq'] = axis_2_spectrum[1]
    df2['z_freq'] = axis_3_spectrum[1]

    t = datetime.now().strftime("%m%d%Y_%H%M%S")
    filename_t = t + '_time.csv'
    filename_f = t + '_freq.csv'
    print(df1)
    #print(df2)
    df1.to_csv(filename_t, index = False)
    df2.to_csv(filename_f, index = False)
    #print('fft end:' + datetime.now().strftime("%m/%d/%Y_%H:%M:%S"))
```

Revision #2
Created 6 June 2024 17:49:31 by Bach_L
Updated 7 June 2024 14:37:15 by Bach_L