Revista Brasileira de Ensino de Física, vol. 42, e20200167 (2020)
www.scielo.br/rbef
DOI: https://doi.org/10.1590/1806-9126-RBEF-2020-0167

Articles

An acquisition system framework for mechanical measurements with Python, Raspberry-Pi and MEMS sensors

Gabriel H. Cassel Barbosa[1], Marcus Varanis[*1], Kelven M. S. Delgado[1], Clivaldo de Oliveira[1]

[1]Universidade Federal da Grande Dourados, Faculdade de Engenharia, Dourados, MS, Brasil

Received on April 30, 2020. Revised on September 08, 2020. Accepted on September 10, 2020.

In this paper, we propose the development of a low-cost acquisition system using Raspberry-Pi 3 and MEMS accelerometers to measure vibrations in mechanical systems. The main objective is to assemble a signals acquisition system easy to handle, of low cost and good accuracy for teaching and industrial purposes. The central idea is that the signals are acquired and processed in the Raspberry-Pi. The Python language and numerical libraries (scipy, numpy and matplotlib) are used for implementation of acquisition and signal processing. This paper proposes the study of vibrations in time and frequency domain. The acquisition system proposed can be easy implemented and the results obtained had good precision (time and frequency domain) and agree with the literature.
Keywords: Acquisition system, Raspberry-Pi, Python, MEMS, Mechanical vibrations.

1. Introduction

Open platform-based microcontrollers, which, in general, use the Arduino and Raspberry-Pi microcontroller as an acquisition system in conjunction with several MEMS sensors to measure vibrations in mechanical systems are now available to a wide range of students, professionals and young researchers, in the most diverse areas of knowledge.

Mechanical vibrations measurement is one of the most important types of measurement applied to engineering today, either for the study itself as modal analysis [1, 2] or even the development of control of dynamical systems based on these studies [3], and we can mention the structural health monitoring of machine components [4], such as rotating elements [5], wind turbines [6], compressors, pumps, fans, among others. Linked to such a study is the use of accelerometers that are sensors of the most variety such as piezoelectric, piezoresistives and capacitives [7].

With the growing availability of low-cost, open source technology in a wide range of devices, associated with ease of use, such as the Arduino and Raspberry-Pi microcontrollers, the application of these devices has been widely used in teaching and scientific development [8]. This work is in line with these new technologies, including a modal study of vibrations using microelectromechanical system accelerometers (MEMS) and the development of high-level Python programming, all together in an affordable Raspberry-Pi based acquisition system.

The Raspberry-Pi is a card-sized microcomputer, whose initial proposal was to bring programming and computer teaching to schools of developing countries. However, the Raspberry-Pi has brought together other open source technologies in which it has enabled applications in various areas of modern physics and engineering. A system based on MEMS accelerometers and Raspberry-Pi can be found in [9], it is a proposal for an urban seismic network in real time in Sicily (Italy). MEMS are already being used in applications such as fall detection systems [10, 11], angle measurement [12] and robot control [13]. Another important part of the Raspberry-Pi development is the possibility of developing applications with cloud storage [14], internet of things (IoT) [15], face recognition [16].

This work also aims to expand the proposals initiated in [17] where the introduction of an easy to use, low cost and good precision signal acquisition system is presented, with didactic objectives for applications in teaching physics and engineering. The same mechanical system proposed in [18] is used here, to compare using Raspberry-Pi, where the performance of two accelerometers of the MEMS type, ADXL335 and MPU6050 with an acquisition system based on the Arduino microcontroller, is evaluated, where the results are obtained to evaluate the dynamic structure response Shear building type with 3 degrees of freedom, excited by a non-ideal source. A long and in-depth study on the use of MEMS accelerometers for the analysis of mechanical vibrations can be seen in [19]. In-depth study of mechanical vibrations and modal analysis with teaching applications using Arduino and MEMS sensors can be seen in [20–22]. In this way, this study proposes the

* Correspondence email address: marcusvaranis@ufgd.edu.br

use of a data acquisition system based on Raspberry-Pi, Python language and MEMS sensors for mechanical measurement applications. As already mentioned, to avoid the complexity of the experiments and especially the high cost involved, this study uses the mini-PC Raspberry-Pi 3 using the Python 3 language and MEMS sensors (ADXL345, MPU6050 and MMA7555). The paper is organized as follows. Some definitions of framework Overview and setup is presented in Section 2. The Experimental Procedure is presented in Section 3. Section 4 describes the experimental results. In Section 5 the paper ends with some final remarks.

2. Framework Overview and Setup

2.1. Raspberry-Pi

The board used in this project is the version Raspberry-Pi 3 model B, which is provided with a BCM2837B0 Broadcom chipset with a 1.4 Ghz clock, 1 GB of RAM memory and a 8 GB Sd card for its removable hard disk memory, there is also a 4.1 Bluetooth, an on board wireless, one HDMI port, four USB port, one Ethernet input and 40 GPIO connection pins for custom hardware connection. It uses 2.5 A and must be adequate connected to a stable 5V power source. All of this concentrated in an 8,56cm × 5,6cm × 2,1cm board. Before using the Raspberry-Pi, it must be properly setup. As it comes with no factory configuration, the operation system (OS) and its installation inside an SD card is essential and for that a personal computer with an SD card reader and internet is needed. This first setup is crucial and the Raspbian operational system comes with all programming software and other programs entirely used in the project. With the SD card already containing the OS and inserted in the device it is also necessary to establish physical connection between Raspberry-Pi and MEMS accelerometer. The digital I2C transmission cables connection is made in the proper Raspberry-Pi's Pins, as shown in Figure 1.

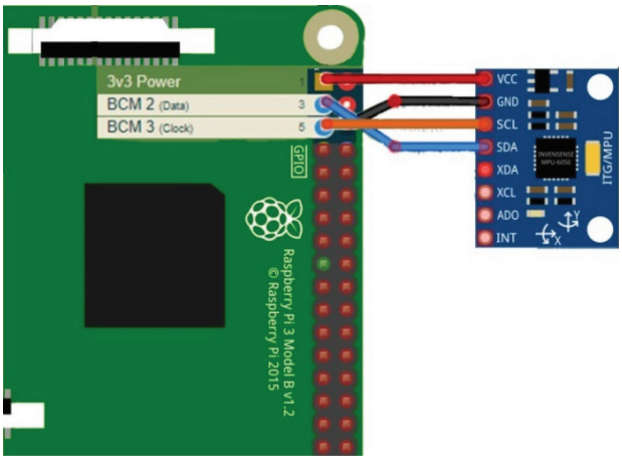

Figure 1: Sensor connection to GPIO ports.

Even with the proper connection, the communication between both devices is yet not possible. Before, the I2C port must be enable inside the Raspberry-Pi configurations so the microprocessor will be able to initialize and use its I2C port. After this, the next step is to install a tool package to detect the accelerometers addresses that are connect to the I2C port. Besides that, in some cases, another configuration must be done inside the folders that define the I2C communication parameters, this is done so the transmission velocity limit is raised using Raspbian system terminal commands.

The last setup is about installing the needed modules for the Python software already incorporated to the OS. Then, the modules used during the project, such as SMBus, datetime and numpy can be installed.

After all the setup steps properly conducted, the Raspberry-Pi can detect the accelerometers and identify its addresses. And the program, written in Python language, will be able to use its modules to configure and collect digital data coming from the accelerometer, according to its respective limitations and factory internal functions further discussed.

2.2. MEMS sensors

In this work, the accelerometers ADXL345, MPU6050 and MMA7455 are used. The ADXL345 is a small, low-power, three-axis accelerometer, with high resolution measurement (13 bits) at 16g (gravity), it has digital output from Analog Devices and selectable measurement ranges in gravities, 2 g, 4 g, 8 g or 16 g, which determine the range of motion, and a fixed sensitivity of 4 mg/LSB. Measuring only 3mm × 5mm × 1mm and with low energy consumption, which is between 40 A to 145 A, and by default the connection via the digital interface I2C and SPI [23]. Its data rate at the output is between 6.25Hz and 3200Hz with a bandwidth between 3.125Hz and 1600Hz. The minimum operating temperature of −40C and the maximum of +85C.

The MPU6050 is the first integrated 6-axis MotionTracking device that combines a 3-axis gyroscope, a 3-axis accelerometer, and a Digital Motion Processor, all on a 4mm × 4mm × 0.9mm plate. The user-programmable accelerometer features scales of 2g, 4g, 8g and 16g, its 16-bit resolution, with a consumption of 500uA in normal operations, and the I2C communication interface. The operating temperature is between −40C to 85C, with bandwidth ranges from 5Hz to 260Hz, and acquisition rate from 10Hz to 520Hz [24].

The MMA7455 is an I2C digital output sensor, low power, capacitive and with signal conditioning and low pass filter. It has a resolution for 3 acceleration ranges, 2g, 4g, 8g. It has a resolution of 10 bits, a consumption of 400uA, operates between the voltage of 2.4V and 3.6V, and works in the temperature range of −40C to 85C. Their bandwidths are 62.5Hz and 125Hz, with their data collection rates of 125Hz and 250Hz [25].

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2020-0167

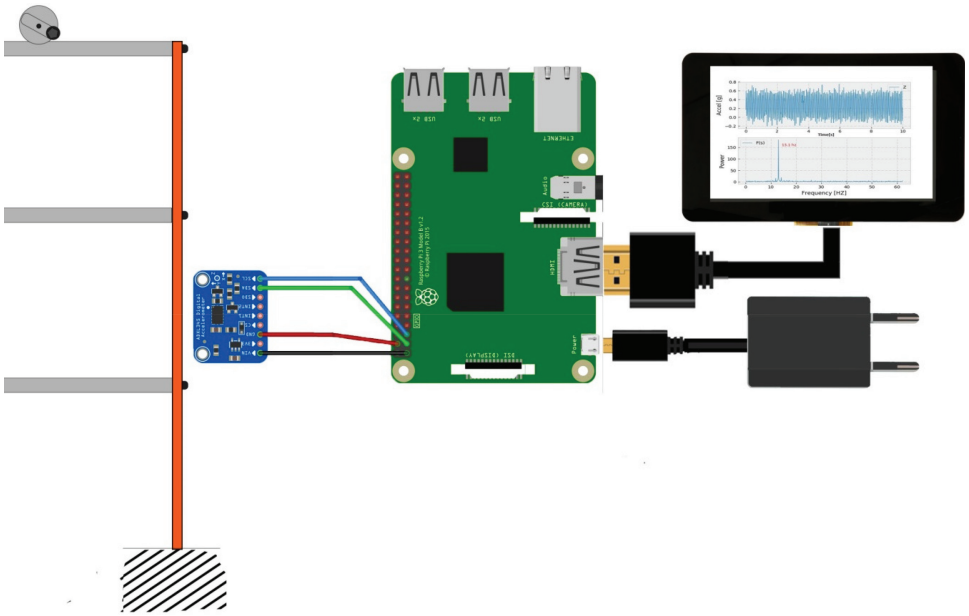

Figure 2: Acquisition system schemes.

2.3. Mechanical system

The acquisition system scheme (Figure 2) that was used is validated from a test bench, already used in [18], Figure 3. As shown in [18] the natural frequencies of the structure are 4.09 Hz, 12.54 Hz and 19.32 Hz. So the experimental procedures and the structure are the same applied previously in a vibration analysis using MEMS and Arduino.

Figure 3: Mechanical system studied (Three story shear building).

2.4. Python programming

The program made in python language must attend the objective of creating an acquisition instrument and provide data storage so the user is able to access, transport and handle the numerical data collected, being free to use it in others software or platforms.

As the python language is vast, there are several useful modules that can manage all the work required by the data acquisition system. Thus, eliminating the need to use more software and others programming languages. Thereby, creating a light program with low computational cost and easy utilization. According to the simple and powerful Raspberry-Pi hardware along the accelerometers.

The most important module is the SMBus, whose utilization aims at controlling the accelerometers by functions which inputs are the accelerometer I2C address, the functions registers and the hexadecimals numbers that define the way the device perform its internal functions which parameters are provided by each accelerometer user guide. All the three show a variety of configurations. Emphasizing that the same module will be used for all the devices, this way there is a possibility of adapting an already completed program so it can work with certain device that offers a I2C digital connection port.

Then, using the module functions, it is possible to send and collect the device data with a proper Python program.

Each function (register) from an accelerometer have its own specific address determined by its user guide in the register map. Not all the functions are configurable, some have only the ability to be readable so can uniquely send information, other functions can only be writable so

can just receive configuration parameters and some have both abilities. It is necessary to read all the information from the manuals, so all functions utilities and modifications are known for appropriate usage. The functions inputs parameters are performed by hexadecimal values. In each user guide configurable register there is a way of modifying internal functions. For that reason, when programming, care must be taken to interpret how to activate and disable what is interesting as output. Finally, the adjusted value for the function configuration must be converted from its binary or decimal base to a hexadecimal base that is introduced in a function as input. Then, the registers are used in the program so the measurement is configured according to the user preferences.

Even though the three accelerometers have different internal functions, the digital communication and the modules utilized are the same. Besides that, the logical sequence conducted by the three is the same.

The discrepancy between the accelerometers is in the configuration of measurements parameters and the data outputted conversion. For each accelerometer, its scales define how the digital data are going to be transformed into decimal base numbers that represent a value of acceleration in a known unity of measure. The more relevant differences are going to be explained next for each accelerometer.

As mentioned before, each accelerometer has different registers, which hexadecimals values, obtained from the user manual, are used in SMBus module functions to change the internal configurations. So, the main differences are the values that represent the register and the parameters that the registers can change. In the ADXL345, the most relevant registers, in hexadecimal base are: $0 \times 2D$, which can change the measurement mode of data; the register $0 \times 2C$, that alter the bandwidth; and the register 0×31, responsible for determining the measuring range. Furthermore, the registers, 0×32 to 0×37, are the output for accelerations data for each axis. For each one of the three axes, there are two outputs, an LSB (Least Significant Bit) and an MSB (Most Significant Bit) output, both having 8 bits each, that need to be converted, as further explained. For the MPU6050, the most significant registers are: $0 \times A1$, capable of setting the bandwidth; the $0 \times 1C$ register, which is responsible for the scale adjusting for the measures; and for configuring the measuring mode, the register $0 \times 6B$. The registers 0×59 to 0×64 are the acceleration output for each axis, also in LSB and MSB, each one having 16 bits of data.

Worth mentioning that this accelerometer also has a gyroscope which can measure angular velocity in three axes. However, neither gyroscope measurements neither internal thermometer are being used in this project, but both can be easily added by implementing the rights registers in the program. Beyond that, this one diverges from the others in his manual, because the configurations are made from a decimal entry to the proper hexadecimal utilized in the program. Different from the others that the configuration of each of the 8-user defined binary bits, which are too switched to hexadecimal values.

Differently from the others, this accelerometer has scale and measure mode configuration in the same register defined by the hexadecimal 0×16. Using the 0×18 register, the bandwidth is chosen, in this case, between 62.5 and 125 Hz. The data output is done through the registers 0×00 to 0×05, which output values have the size of 10 bits and each axis present data in LSB and MSB. In the same way as the ADXL345, the configuration parameters must be determined from a binary base and then converted to the hexadecimal base, the correct input in the functions from the SMBus module.

2.5. Data acquisition

Having all the register and the proper configurations parameters, the write functions inside the SMBus module are used to send and determine which scale, bandwidth and measure mode will be assigned. After that, a loop made with the output acceleration data registers is used to gather the data. This loop was based on another used in a project that collected data with Arduino and MEMS accelerometers [17].

To deal with time, the datetime class from the datetime module was chosen due to its accuracy and easy handle of operations in the microsecond resolution. Thus, the loop compares the times between the difference of start from the previous acquisition and the current time so when the period from the user determined frequency is reached a new acquisition is started and the process of comparison initiates again. This loop keeps gathering data until the user determined duration is reached, stopping the measurement.

When using datetime module in time comparison operations, attention should be taken in the fact that only objects of the same kind from these modules can be compared. For that reason, frequency and measure duration inserted by the user, which are "string" class objects, must be converted to datetime class objects, so both can be compared.

In each data gathering, output functions for the three axes are used together with hexadecimal registers and the IC accelerometers registers. Is now that the read functions from the SMBus module are applied with the output data registers for each axis in LSB and MSB and all data from the record is stored in a list, in which each line is a samples and the six columns are the values obtained from the registers. Beyond that, the time when each sample was gathered is stored in another list so it can be displayed with the acceleration to the user. Much importance was given to use low processing memory from the Raspberry-Pi. Therefore, real time data is no shown to the user as this operation would waste memory. Thus, all data is only stored by the list method "append", which add elements to the acceleration and

time lists each time data is collect in the loop so computational cost during measurement is reduced. This way, lower is the chance of a gathering take more time than the period required by the frequency inserted by the user.

2.6. Data conversion

After the acquisition, all the acceleration is inside a large list and inside each column there is a list containing both LSB and MSB data for each axis. The next step is to apply a function defined inside the program to make the proper conversion for the accelerometer out- put data. This is done for the three axis and for all the values inside the lines which are the samples recorded in each loop and stored in the large acceleration list.

Each accelerometer has a different conversion function because theirs outputs have different sizes. While the ADXL345 and MMA7455 data in 10 bits, the MPU6050 sends it in 16 bits. Thus, each acceleration output in LSB must be combined with the MSB out- put and then multiplied by a scale factor so the output values are turned in to decimal values in the standard gravity acceleration unit.

The conversion function uses python logic operators to manipulate and combine binary data. Depending on the output value a function is used so is necessary one for the 8 bits output and another for the 16 bits. Then, the acceleration list that holds the collected data from the loop is converted into a list with only three columns, one for each axis.

After combined, the values must be multiplied by the scale factors, which transforms data to the gravity acceleration units. The factors change for each scale and output measured so they must be obtained inside the accelerometers devices documentation.

A code implementation allows the user to choose the measurement scale, so the adequate factor is adjusted for the last conversion to the proper magnitude. After that, the values can be easily transformed to the international acceleration unity system (m/s^2) by a simple multiplication. Thus, the final values are ready to be stored utilized outside the measurement device program.

2.7. Data storage

An easy and well known way to extract and save this already converted data is by creating a cvs ("Comma Separated Value") format file, in which can be read by the most common and free spreadsheet editors, already incorporated in the Raspberry-Pi operational system.

For this purpose, a csv module must be utilized and the data inside the list be converted to string class when writing the new file. The file name takes the actual date and time when the creation is being done to make easier to handle sequential measurements. Beyond that, a loop is used so all lines and columns from the list are placed inside the new archive which is closed and saved when the loop reaches its end.

2.8. Signal processing

In the acquisition and experimental signals for the analysis of mechanical vibrations, through accelerometers, it is ideal to analyze not only their representation in the time domain, but also their response in the frequency domain. For the response in the frequency domain, the Fast Fourier Transform (FFT) algorithm was used. In this work, it was used the fftpack module from the scipy library.

Another important implementation was the development of time-frequency analysis through Short time Fourier Transform (STFT), in order to quantify the change in the frequency of a non-stationary signal over time. In this work, it was used the module signal.stft from the scipy signal processing library. All graphics generated by the acquisition system are made with the Matplolib module from the same library.

In order to produce the FFT and STFT it is necessary a time step, that is the sample rate of the measurement. During the data gathering loop, the code compares the previous time that data was collected with real time provided by the Raspberry Pi to assure that the right period of time determined by the user is followed. However, the time collected from the microcomputer comes with its own part of error and commands used in the code take some time to be processed by the computer. Therefore, sample rate is asynchronous and each measurement has different errors in time period that depends of computational processing. As time sample is not uniform, the value used for FFT and STFT is a mean, provided by the division of total gathering operation time and total samples. It is a very complex task to obtain synchronous rates using such a simple wiring connection and depending on computers, even more time accuracy must be pursued in future research.

3. Experimental Procedure

For the resonance condition, using an unbalanced motor, whose vibration frequency is defined by the voltage supplied which is described by the Figure 4 graph, receiving power from a controllable voltage power source. The data were collected using the measuring instruments for the exciting motor in the frequencies of 4.2Hz, 13.12Hz and 20.4Hz, corresponding respectively to a voltage of 3V, 6V and 9V. All measurements were made within a given interval of 10 seconds, enough time to observe the vibratory nature of the structure in resonance. In order to demonstrate the application of signals with non-stationary characteristics through the STFT, a second experiment conditioned the structure with a motor that started from zero up to the frequency of 20.4 Hz with its corresponding 9V of voltage in the power source of the motor. Then, the same 10 seconds interval applied in this procedure for all accelerometers. The location of the accelerometer sensor is presented in Figure 5.

In the experiment, all three accelerometers must be positioned in the right place, the placement is in the

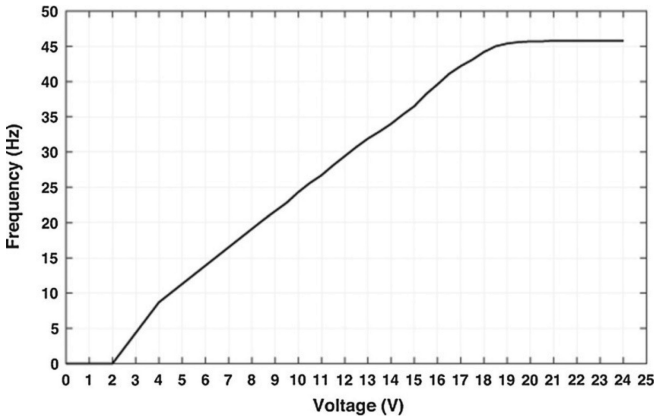

Figure 4: Voltage vs frequency curve of the motor used.

Figure 5: Location of the accelerometer sensor.

lower part on the side of the highest plate that made the structure as shown in Figure 6. The Raspberry-Pi was connected to the accelerometer and the necessary display controller and its power source Figure 6.

Each type of measurement must be evaluated prior to determine the parameters chosen to suite the limits

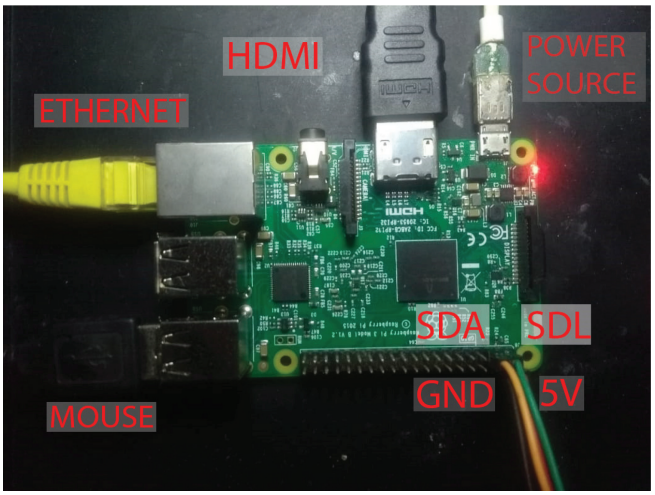

Figure 6: Raspberry-Pi 3 and connections during experiment.

of the measurement instrument and the magnitude of the evaluated phenomenon. Then, for the measurement of the excited structure at a maximum of 20.4 Hz, the minimum quality requires a sampling frequency of 40.5 Hz, twice the original signal frequency of the structural excitation, following the Nyquist Sampling Theorem for the correct construction of the signal by the Fast Fourier Transform. As each accelerometer presents different bandwidth configurations, different sampling frequencies are used as needed for the vibration acceleration signal of the structure.

First, for the MPU6050 accelerometer, all collections were made with the measuring instrument set to the same bandwidth of 94 Hz and a 188 Hz sampling frequency. Being 94 Hz the appropriate bandwidth for the maximum rated excitation of 20.4 Hz.

Using the MMA7455, the instrument settings for the measurements excited by 4.2 Hz and 13.12 Hz were 62.5 Hz for a bandwidth and 125 Hz for a sampling frequency. For the 20.4 Hz excitation, a bandwidth and sampling frequency respectively set to 125 Hz and 250 Hz.

For the ADXL345 configuration, the 4.2 Hz and 20.4 Hz frequency excitation measurements were made with 50 Hz bandwidth and 100 Hz sample collection frequency. Then for 13.12 Hz excitation, 100 Hz bandwidth and 200 Hz sampling frequency were selected for the acceleration acquisition.

The experimental procedure followed all the instruction to be as close as the procedures of vibration measuring conducted in the previous works using the same mechanical structure. Thus, more details in how to correctly repeat the experiment can be found in [18], which results will be compared to the obtained in this work.

4. Results

As the results show in Figures 7, 8 and 9 for the measure of the mechanical system in a resonance condition,

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2020-0167

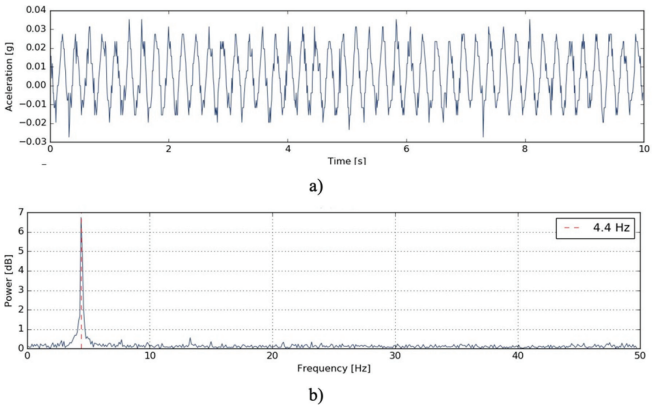

Figure 7: Signal measured when the structure is excited by a frequency of 4.2 Hz using the ADXL-345 accelerometer: a) Time history and b) Fast Fourier Transform.

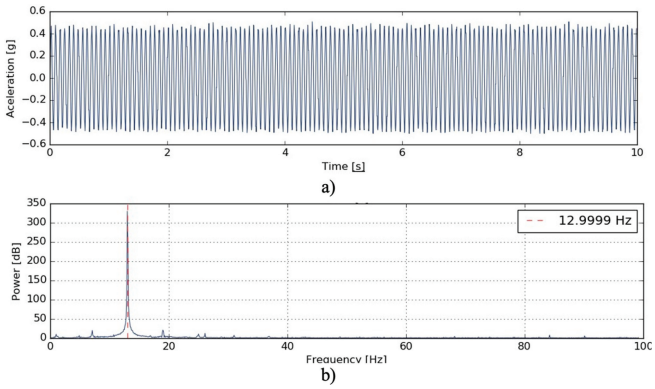

Figure 8: Signal measured when the structure is excited by a frequency of 12.99 Hz using the ADXL-345 accelerometer: a) Time history and b) Fast Fourier Transform.

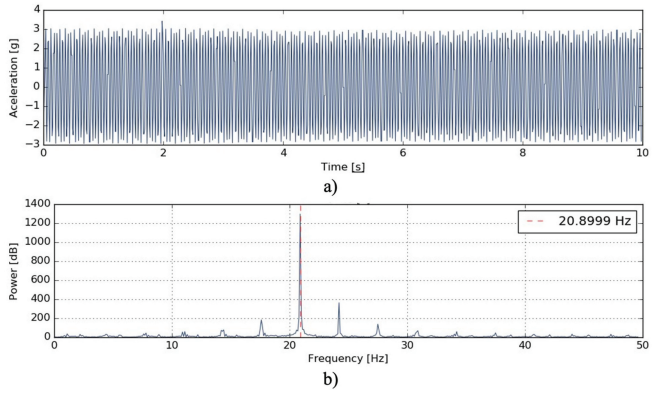

Figure 9: Signal measured when the structure is excited by a frequency of 20.89 Hz using the ADXL-345 accelerometer: a) Time history and b) Fast Fourier Transform.

the FFT was capable of provide a peak that approaches the frequency expected from the excitation of the unbalanced motor in the three different vibration frequencies (4.2 Hz, 12.99 Hz, 20.89 Hz) provided by the motor using the ADXL345. The other accelerometers presented similar results of acceleration and the same peak for the frequencies in this condition. This peak is observed and comparable to previous works and matches with the results obtained for the measuring instrument from this work in the same experimental procedure. All the comparisons are in evidence in Table 1.

For the starting motor experiment, the results at Figures 10, 11 and 12, the STFT shows that the vibration frequency increases over the 10 seconds interval. It is possible to notice, in all the analyzes made with STFT, the characterization of the 3 natural frequencies of the structure, during the excitation through the motor start, non-stationary regime.

Table 1: Comparison table of the results of this work with the previous of [17] and [18].

This work			
	ω_1	ω_2	ω_3
MPU6050	4.1 Hz	13.09 Hz	20.89 Hz
MMA7455	4.2 Hz	12.99 Hz	20.89 Hz
ADXL345	4.4 Hz	12.99 Hz	20.89 Hz
Varanis 2017			
Analytical	4.09 Hz	12.54 Hz	19.32 Hz
ADXL335	4.08 Hz	12.94 Hz	20.26 Hz
MPU6050	4.10 Hz	12.84 Hz	20.25 Hz
Varanis 2018			
Analytical	4.09 Hz	12.54 Hz	19.32 Hz
FEM	4.75 Hz	14.69 Hz	24.36 Hz
DeltaTron	4.028 Hz	12.94 Hz	20.26 Hz
MPU6050	4.028 Hz	12.94 Hz	20.26 Hz
ADXL345	4.028 Hz	12.94 Hz	20.26 Hz
ADXL335	4.028 Hz	12.94 Hz	20.26 Hz
Difference (Calculated in relation to Analytical results – Varanis 2018)			
MPU6050	0,24%	4.39%	8.13%
MMA7455	2.69 %	3.59%	8.13%
ADXL345	7.58%	3.59%	8.13%

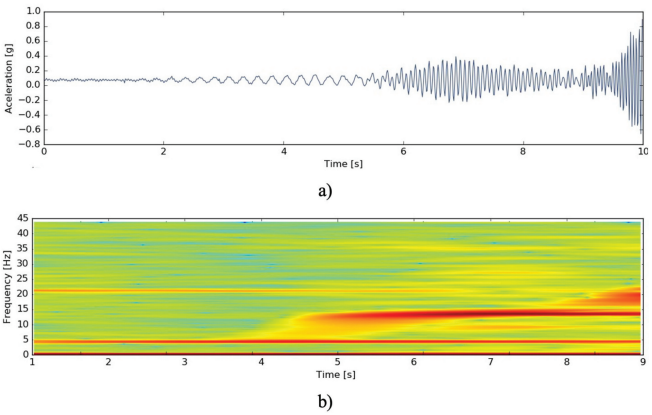

Figure 10: Signal measured (motor run-up 0-9v) measurement with MPU6050: a) Time history and b) Short Time Fourier Transform.

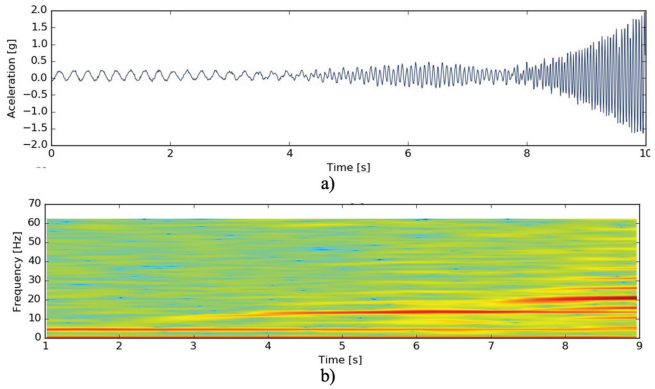

Figure 11: Signal measured (motor run-up 0-9v) measurement with MMA7455: a) Time history and b) Short Time Fourier Transform.

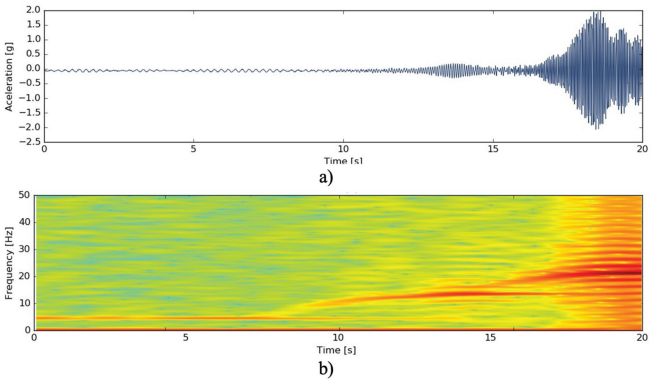

Figure 12: Signal measured (motor run-up 0-9v) measurement with ADXL345: a) Time history and b) Short Time Fourier Transform.

5. Final Remarks

This article presents an expansion of previous works started in [17, 18], which proposed to measure mechanical vibrations using the Arduino microcontroller and low-cost sensors for educational purposes. The results presented in this work show good precision of the acquisition system in the domain of time and frequency, when compared to other instruments and analytical analyzes for the same mechanical structure in two conditions of vibration. Its low cost and small size have proven to be an easy-to-use resource in modal analysis and mechanical vibration applications. One of the great advantages of the proposed system is the fact that the acquisition, processing, and storage of the signals takes place in the Raspberry-Pi itself, in an automated way without the need for external software to export the registered data. With respect to signal processing, the system can perform analysis via FFT and time-frequency analysis using STFT, which allows the analysis of non-stationary signals. The Python language was also shown to be suitable for project implementation due to the ease of communication with the sensors and the construction of data acquisition and signal processing mechanisms using the mentioned libraries. The sensors used showed results in accordance with the literature, as shown in Table 1. Due to the low cost, ease of customization and presented results, the presented framework is very useful for applications in undergraduate and graduate courses graduation in physics and engineering, in addition to presenting the possibility of being used in a wide range of applications, from teaching and research to even industrial.

For accelerometers used, there is a maximum data rate, a maximum sample frequency, that the connections can receive and sending between the two devices that make up the instrument. This is because the digital I2C communication between the accelerometers and the Raspberry-Pi is simple but limited in the number of bytes that can be transferred per second. Therefore, that one used for the measuring instrument is suitable only in the case of acceleration measurements whose sampling rate is below the frequency specified for each accelerometer.

Acknowledgments

The authors acknowledge the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) for the support.

Appendix

ADXL345

```
import smbus2 as smbus
#import smbus
from datetime import datetime, timedelta
import time
from numpy import*

bus = smbus.SMBus(1)  # Defines Bus as object of class
      Smbus
address = 0x53    # Accelerometer hexadecimal address

# PARAMETERS AND RECORDS IN HEXADECIMAL
      ADXL345

ACCEL_2G  = 0x00 #RANGE
ACCEL_4G  = 0x01
ACCEL_8G  = 0x02
ACCEL_16G = 0x03

SCALE_MULTIPLIER_2G = 4/1024
     # MULTIPLICATORS FOR G
SCALE_MULTIPLIER_4G = 8/1024
SCALE_MULTIPLIER_8G = 16/1024
SCALE_MULTIPLIER_16G = 32/1024

BW_RATE_1600HZ = 0x0F
BW_RATE_800HZ = 0x0E
BW_RATE_400HZ = 0x0D
BW_RATE_200HZ = 0x0C
BW_RATE_100HZ = 0x0B
BW_RATE_50HZ = 0x0A
BW_RATE_25HZ = 0x09
BW_RATE_12_5HZ = 0x08
```

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2020-0167

```
BW_RATE_6_25HZ = 0x07
BW_RATE_3_13HZ = 0x06
BW_RATE_1_56HZ = 0x05
BW_RATE_0_78HZ = 0x04
BW_RATE_0_39HZ = 0x03
BW_RATE_0_20HZ = 0x02
BW_RATE_0_10HZ = 0x01
BW_RATE_0_05HZ = 0x00

SCALE_REGISTER = 0x31 #RANGE CONTROL
BW_REGISTER = 0x2C # CONTROL BANDWIDTH
MODE_REGISTER = 0x2D # CONTROL THE
    MEASUREMENT MODE
MEASURE_MODE = 0x08 # MEASURE MODE

# -- Functions --

def convert_LSBandMSB(raw_val): # CONVERTS DATE
    TO 10 BITS
  convert_val = (raw_val [1] \& 0x03) * 256 + raw_val [0]
  if convert_val > 511:
      convert_val -= 1024
  return convert_val

def option_scale(SCALE):
  if scale == int(2):
    return [ACCEL_2G,SCALE_MULTIPLIER_2G]
  elif scale == int(4):
    return [ACCEL_4G,SCALE_MULTIPLIER_4G]
  elif scale == int(8):
    return [ACCEL_8G,SCALE_MULTIPLIER_8G]
  elif scale == int(16):
    return [ACCEL_16G,SCALE_MULTIPLIER_16G]

def option_banda(banda):
  if banda == int(0.05):
    return BW_RATE_0_05HZ
  elif banda == int(0.10):
    return BW_RATE_0_10HZ
  elif banda == int(0.20):
    return BW_RATE_0_20HZ
  elif banda == int(0.39):
    return BW_RATE_0_39HZ
  elif banda == int(0.78):
    return BW_RATE_0_78HZ
  elif banda == int(1.56):
    return BW_RATE_1_56HZ
  elif banda == int(3.13):
    return BW_RATE_3_13HZ
  elif banda == int(6.25):
    return BW_RATE_6_25HZ
  elif banda == int(12.5):
    return BW_RATE_12_5HZ
  elif banda == int(25):
    return BW_RATE_25HZ
  elif banda == int(50):
    return BW_RATE_50HZ
  elif banda == int(100):
    return BW_RATE_100HZ
  elif banda == int(200):
    return BW_RATE_200HZ
  elif banda == int(400):
    return BW_RATE_400HZ
  elif banda == int(800):
    return BW_RATE_800HZ
  elif banda == int(1600):
    return BW_RATE_1600HZ

# - USER INPUTS

RANGE_XG=int()
scale = int(input(''Scale [2,4,8,16]:''))
RANGE_XG = option_scale(scale)

BANDW_XHZ=int()
banda = int(input(''Bandwidth [0.05, 0.1, 0.2, 0.39, 0.78,
    1.56, 3.13, 6.25, 12.5, 25, 50, 100, 200, 400, 800,
    1600]:''))
BANDW_XHZ = option_banda(banda)

freqInput = input(''Measuring frequency[Hz]:'')

#ene=input(''2^(?):'')    #nsample
#nsamp=2**int(ene)        #nsample

minutes = input(''Measurement Duration[m]:'')
seconds = input(''Measurement Duration[s]:'')
duration = timedelta(0,int(seconds),0,0,int(minutes))

time.sleep(1)

# MEASUREMENT CONFIGURATION
bus.write_byte_data(address, MODE_REGISTER,
    MEASURE_MODE)
alem=bus.read_byte_data(address, MODE_REGISTER)
if alem == MEASURE_MODE:
  print(''CORRECTLY CONFIGURED MODE'')
time.sleep(0.2)
bus.write_byte_data(address, BW_REGISTER,
    BANDW_XHZ)
allm=bus.read_byte_data(address, BW_REGISTER)
if allm == BANDW_XHZ:
  print(''BANDWIDTH CORRECTLY CONFIGURED'')
time.sleep(0.2)
bus.write_byte_data(address, SCALE_REGISTER,
    RANGE_XG[0])
alum=bus.read_byte_data(address, SCALE_REGISTER)
if alum == RANGE_XG[0]:
  print(''SCALE CORRECTLY CONFIGURED'')

print(str(RANGE_XG[1]))

time.sleep(1)

# TRANSFORM THE STRING FREQUENCY INPUT
    TO DATETIME
milsec = float(1000.0/float(freqInput))
if milsec==1000:
  period=timedelta(0,1)
elif milsec%1==0:
  period=timedelta(0,0,0,int(milsec))
else:
  period=timedelta(0,0,(milsec-int(milsec))*1000,
      int(milsec))

# CREATING LIST WHERE DATA WILL BE
    RECORDED WITH APPEND
tempo=[]
accdata=[]
t=0
time.sleep(1)

# start control

while True:
  init = input(''Start Data Acquisition (y/n):'')
  if init == ''y'':
    break
  else:
    pass

# LOOP DATA ACQUISITION
print('Start of Data Acquisition')
previousMillis = datetime.now()
start = datetime.now()
endloop=datetime.now()+duration
while datetime.now().time() <endloop.time():
  currentMillis = datetime.now()
```

```
  if (currentMillis - previousMillis >= period):
     previousMillis = datetime.now()
     accdata.append([[bus.read_byte_data(address,
0x32),bus.read_byte_data(address, 0x33)],[bus.
      read_byte_data(address,
0x34),bus.read_byte_data(address, 0x35)],[bus.
      read_byte_data(address,
0x36),bus.read_byte_data(address,0x37)]])
     #accdata.append([[bus.read_byte_data(address,
0x36),bus.read_byte_data(address, 0x37)]])
     tempo.append(datetime.now())

end=datetime.now()
acqt = end-start
print('|------- End of Data Acquisition ------- |')

# CONVERSION OF INPUTS IN LSB AND MSB FOR
     GRAVITY

for i in range(len(tempo)):
  for j in range(0,3):
    accdata[i][j] = convert_LSBandMSB(accdata[i][j]) *
      RANGE_XG[1]

# FILE OPENING
ttime=[]
opentime = datetime.now()
nome = opentime.strftime('ADXL_Data'+'(%d|%m|%Y_%
     H: %M: %S)')
saida = open(nome+''.csv'',''w'')

for i in range(len(tempo)):
  saida.write(str(i+1)+'';'')
       # MEASUREMENT NUMBER
  saida.write(str(tempo[i].time())+'';'  '')       # TIME
  saida.write(str((tempo[i]-tempo[0]).total_seconds())+'';'')
       # Time from start to
measurement
  saida.write(str(accdata[i][0])+'';'')
       # Accelerometer X
  saida.write(str(accdata[i][1])+'';'')
       # Accelerometer y
  saida.write(str(accdata[i][2])+'';'')
       # Accelerometer z
  saida.write(str(acqt)+''\n'')
  ttime.append((tempo[i]-tempo[0]).total_seconds())
saida.close()

'''---------- Important definitions ----------'''

acc=transpose(accdata)
sig=acc[2]
time_step=acqt.total_seconds()/len(sig)
Fs=1/time_step

'''-------- Starting to calculate FFT --------'''

from scipy import fftpack

sample_freq = fftpack.fftfreq(sig.size, d=time_step)
     # generates sampling frequencies
sig_fft = fftpack.fft(sig)              # calculates the fast
     Fourier transform
pidxs = where(sample_freq> 0)                 # Only the
     positive part of the spectrum seràutilizada.
freqs = sample_freq[pidxs]
power = abs(sig_fft)[pidxs]

print (''| Fininish FFT'')

'''--- Find the frequency according to the peak ---'''

N_freq=int(input('======== Enter number of
     frequencies to find:'))

xcoords=zeros((N_freq))
DT=int(len(freqs)/N_freq)

for i in range(N_freq):
  fr=freqs[(i)*DT: DT*(i+1)]
  xcoords[i]=fr[argmax(power[(i)*DT: DT*(i+1)])]

'''---------- Name to be identified ----------'''

dataname=nome+''.csv''
figname = opentime.strftime('ADXL_Figure_'+'(%d|%m|%
     Y_%H: %M: %S)'+'.png')
specname=opentime.strftime('ADXL_Specgram_'+'(%d|%m
     |%Y_%H: %M: %S)'+'.png')

'''---------- Gràfico ----------'''

import matplotlib.pyplot as plt

plt.figure(1,figsize=[12,6])
plt.style.use('classic')

plt.subplot(2, 1, 1)
plt.plot(ttime,sig,color='#2f4875',linewidth=0.8)
plt.xlabel('Time [s]]')
plt.ylabel('Acceleration [g]')

plt.subplot(2, 1, 2)
plt.plot(freqs, power,color='#2f4875',linewidth=0.8)
plt.xlabel('Frequency [HZ]')
plt.ylabel('Power [dB]')
for xc in xcoords:
  plt.axvline(x=xc,linestyle='--
',linewidth=1,color='#FF4040',label=str(round(xc,4))
     +'Hz')
plt.legend()
plt.grid(True)

plt.savefig(figname)
plt.show(block=False)
plt.pause(5)
plt.close()

plt.figure(2,figsize=[12,6])
plt.style.use('classic')

plt.subplot(2,1,1)
plt.plot(ttime,sig,color='#2f4875',linewidth=0.8)
plt.xlabel('Time [s]')
plt.ylabel('Acceleration [g]')
plt.subplot(2,1,2)
powerSpectrum,frequenciesFound,time,imageAxis=plt.
     specgram(sig,Fs=Fs,noverlap=2
55,cmap='Spectral')
plt.xlabel('Time [s]')
plt.ylabel('Frequency [Hz]')
plt.savefig(specname)
plt.show(block=False)
plt.pause(5)
plt.close()

print (''| ------- Figure Saved ------- |'')

'''----------Dropbox----------'''

entrada=(figname,dataname,specname)

while True:
  resp=input('Save in dropbox (y/n):\t')

  if resp=='y':
     name_data=input('Enter data file name:\t')
     name_fig=input('Enter figure file name:\t')
     name_spec=input('Enter specgram file name: \t')
```

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2020-0167

```
    saida = ('/E.R.A/'+name_fig+opentime.strftime('(%d
      |%m|%Y_%H: %M:
%S)')+'.png','/E.R.A/'+name_data+opentime.strftime('(%
      d|%m|%Y_%H: %M:
%S)')+'.csv','/E.R.A/'+name_spec+opentime.strftime('(%d
      |%m|%Y_%H: %M:
%S)')+'.png')

    print('| --- Start file writing in Dropbox --- |')

    import dropbox


    class TransferData:
      def __init__(self, access_token):
        self.access_token = access_token

      def upload_file(self, file_from, file_to):
        """upload a file to Dropbox using API v2
        """
        dbx = dropbox.Dropbox(self.access_token)

        with open(file_from, 'rb') as f:
          dbx.files_upload(f.read(), file_to)


    def main():
      access_token =
'bCzFzRUXC2AAAAAAAAAAeA_XpUcnbr7s6CklT0oFYc
      CxaJ11xnpDvDytIcEg8hrt'
      transferData = TransferData(access_token)

      for i in range(3):
        file_from = entrada[i]
        file_to = saida[i]
        # API v2
        transferData.upload_file(file_from, file_to)


    if __name__ == '__main__':
      main()
    break
  elif resp=='n':

    break
  else:
    pass

print ('| ------- Completely Finish ------- |')
```

MMA7455

```
#!/usr/bin/env python 3.5.3

import smbus2 as smbus
from datetime import datetime, timedelta
import time
from numpy import*


bus = smbus.SMBus(1) # Bus for Revision 2 boards
address = 0x1d# Sensor i2c address

# PARAMETERS AND RECORDS IN HEXADECIMAL
      MMA7455

ACCEL_2G = 0x05  # measurement mode and scales
ACCEL_4G = 0x59
ACCEL_8G = 0x41

SCALE_MULTIPLIER_2G = 4/1024
SCALE_MULTIPLIER_4G = 8/1024
SCALE_MULTIPLIER_8G = 16/1024

MODE_SCALE_REGISTER = 0x16   #mode and scale
CONTROL1 = 0x18    # bandwidth control

BW_RATE_62_5HZ = 0x00   # bandwidth
BW_RATE_125HZ = 0x80

scale_mult=0

# FUNCTIONS

def LSBandMSB_10bits(data):
  value = (data[1] \& 0x03) * 256 + data[0]
  if value > 511:
    value -= 1024
  return value

def option_scale(scale):
  if scale == int(2):
    return [ACCEL_2G,SCALE_MULTIPLIER_2G]
  elif scale == int(4):
    return [ACCEL_4G,SCALE_MULTIPLIER_4G]
  elif scale == int(8):
    return [ACCEL_8G,SCALE_MULTIPLIER_8G]


def option_banda(banda):
  if banda == int(62):
    return BW_RATE_62_5HZ
elif banda == int\eqref{GrindEQ__125_}:
    return BW_RATE_125HZ

# USER INPUTS

RANGE_XG=int()
scale = int(input(''Scale [2,4,8]:''))
RANGE_XG = option_scale(scale)

BANDW_XHZ=int()
banda = int(input(''Bandwidth [62.5, 125]:''))
BANDW_XHZ = option_banda(banda)

freqInput = input(''Measuring frequency[Hz]:'')

#ene= input(''2^(?):'')  #nsample
#nsamp=2**int(ene

minutes = input(''Measuring duration[m]:'')
seconds = input(''Measuring duration[s]:'')
duration = timedelta(0,int(seconds),0,0,int(minutes))

# MEASUREMENT CONFIGURATION
bus.write_byte_data(address, CONTROL1, BANDW_XHZ)
allm=bus.read_byte_data(address, CONTROL1)
if allm == BANDW_XHZ:
  print(''BANDWIDTH CORRECTLY CONFIGURED'')
time.sleep(0.2)
bus.write_byte_data(address, MODE_SCALE_REGISTER,
      RANGE_XG[0])
alum=bus.read_byte_data(address,
      MODE_SCALE_REGISTER)
if alum == RANGE_XG[0]:
  print(''CORRECTLY CONFIGURED SCALE AND
      MODE'')

print(str(RANGE_XG[1]))
time.sleep(1)

# TRANSFORM THE STRING FREQUENCY INPUT
      TO DATETIME
milsec = float(1000.0/float(freqInput))
if milsec==1000:
  period=timedelta(0,1)
elif milsec%1==0:
  period=timedelta(0,0,0,int(milsec))
else:
  period=timedelta(0,0,(milsec-int(milsec))*1000,int
      (milsec))

# CREATING LIST where data will be recorded with
      append
tempo=[]
```

```
accdata=[]
t=0
time.sleep(1)

N_freq=int(input('========== Enter the number of
    frequencies to find:'))

while True:
  init = input(''Start Data Acquisition (y/n):'')
  if init == ''y'':
    break
  else:
    pass

# LOOP DATA ACQUISITION
previousMillis = datetime.now()
start= datetime.now()
endloop=datetime.now()+duration
while datetime.now().time() < endloop.time():
  currentMillis = datetime.now()
  if (currentMillis - previousMillis >= period):
    previousMillis = datetime.now()
    accdata.append([[bus.read_byte_data(address,0x00),bus.
      read_byte_data(address,
0x01)],[bus.read_byte_data(address, 0x02),bus.
      read_byte_data(address,
0x03)],[bus.read_byte_data(address, 0x04),bus.
      read_byte_data(address, 0x05)]])
    tempo.append(datetime.now())

end=datetime.now()
acqt = end-start
print('End of Data Acquisition')

# CONVERSION OF INPUTS IN LSB AND MSB FOR
    GRAVITY

for i in range(len(tempo)):
  for j in range(0,3):
    accdata[i][j] = LSBandMSB_10bits(accdata[i][j]) *
      RANGE_XG[1]

# CSV FILE OPENING
ttime=[]
opentime = datetime.now()
nome = opentime.strftime('MMA_Data'+'(%d|%m|%Y_%H
    : %M: %S)')
saida = open(nome+''.csv'',''w'')

for i in range(len(tempo)):
  saida.write(str(i+1)+'';'') # MEASUREMENT
      NUMBER
  saida.write(str(tempo[i].time())+'';'') # TIME
  saida.write(str((tempo[i]-tempo[0]).total_seconds())+'';'')
      # Time from start to
measurement
  saida.write(str(accdata[i][0])+'';'') # Accelerometer X
  saida.write(str(accdata[i][1])+'';'') # Accelerometer y
  saida.write(str(accdata[i][2])+'';'') # Accelerometer z
  saida.write(str(acqt)+''\n'')
  ttime.append((tempo[i]-tempo[0]).total_seconds())
saida.close()

'''------- Important definitions -------'''

acc=transpose(accdata)
sig=acc[2]
time_step=acqt.total_seconds()/len(sig)
Fs=1/time_step

'''------- Starting to calculate FFT -------'''

from scipy import fftpack

sample_freq = fftpack.fftfreq(sig.size, d=time_step)
    # generates sampling frequencies
sig_fft = fftpack.fft(sig)          # calculates the fast
    Fourier transform
pidxs = where(sample_freq > 0)          # Only the
    positive part of the spectrum
will be used.
freqs = sample_freq[pidxs]
power = abs(sig_fft)[pidxs]

print (''| Fininish FFT'')

'''------- Find frequency peak -------'''

xcoords=zeros((N_freq))
DT=int(len(freqs)/N_freq)

for i in range(N_freq):
  fr=freqs[(i)*DT: DT*(i+1)]
  xcoords[i]=fr[argmax(power[(i)*DT: DT*(i+1)] ) ]

'''------- Figure name -------'''

dataname=nome+''.csv''
figname = opentime.strftime('MMA_Figure_'+'(%d|%m| %
    Y_%H: %M: %S)'+'.png')
specname=opentime.strftime('MMA_Specgram_'+'(%d |%m
    | %Y_%H: %M: %S)'+'.png')

'''-------Grafic-------'''

import matplotlib.pyplot as plt

plt.figure(1,figsize=[12,6])
plt.style.use('classic')

plt.subplot(2, 1, 1)
plt.plot(ttime,sig,color='#2f4875',linewidth=0.8)
plt.xlabel('Time [s]')
plt.ylabel('Acceleration [g]')

plt.subplot(2, 1, 2)
plt.plot(freqs, power,linewidth=0.8)
plt.xlabel('Frequency [HZ]')
plt.ylabel('Power')
for xc in xcoords:
  plt.axvline(x=xc,linestyle='--
',linewidth=1,color='#FF4040',label=str(round(xc,4))
    +'Hz')
plt.legend()
plt.grid(True)

plt.savefig(figname)
plt.show(block=False)
plt.pause(5)
plt.close()

plt.figure(2,figsize=[12,6])
plt.style.use('classic')

plt.subplot(2,1,1)
plt.plot(ttime,sig,color='#2f4875',linewidth=0.8)
plt.xlabel('Time [s]')
plt.ylabel('Acceleration [g]')
plt.subplot(2,1,2)
powerSpectrum,frequenciesFound,time,imageAxis=plt.
    specgram(sig,Fs=Fs,noverlap=255,cmap='Spectral')
plt.xlabel('Time [s]')
plt.ylabel('Frequency [Hz]')
plt.savefig(specname)
plt.show(block=False)
plt.pause(5)
plt.close()

print (''| ------- Figures Saved ------- |'')
```

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2020-0167

```
'''——————— Dropbox ———————'''

entrada=(figname,dataname,specname)

while True:
  resp=input('Save in dropbox (y/n):\t')

  if resp=='y':
     name_data=input('Enter data file name:\t')
     name_fig=input('Enter figure file name:\t')
     name_spec=input('Enter specgram file name: \t')
     saida = ('/E.R.A/'+name_fig+opentime.strftime('(%d
       |%m|%Y_%H: %M:
%S)')+'.png','/E.R.A/'+name_data+opentime.strftime('(%
       d|%m|%Y_%H: %M:
%S)')+'.csv','/E.R.A/'+name_spec+opentime.strftime('(%d
       |%m|%Y_%H: %M:
%S)')+'.png')

     print('| ——— Start file writing in Dropbox ——— |')

     import dropbox


     class TransferData:
       def __init__(self, access_token):
         self.access_token = access_token

       def upload_file(self, file_from, file_to):
         """upload a file to Dropbox using API v2
         """

         dbx = dropbox.Dropbox(self.access_token)

         with open(file_from, 'rb') as f:
           dbx.files_upload(f.read(), file_to)


    def main():
      access_token =
'bCzFzRUXC2AAAAAAAAAAeA_XpUcnbr7s6CklT0oFYc
       CxaJ11xnpDvDytIcEg8hrt'
      transferData = TransferData(access_token)

      for i in range(3):
        file_from = entrada[i]
        file_to = saida[i]
        # API v2
        transferData.upload_file(file_from, file_to)


   if __name__ == '__main__':
       main()
   break
elif resp=='n':

   break
else:
   pass

print ('| ——————— Completely Finish ——————— |')
```

MPU6050

```
#!/usr/bin/env python 3.6

import smbus2 as smbus
from datetime import datetime, timedelta
import time
from numpy import*

bus = smbus.SMBus(1) # Defines Bus as object of class
      Smbus
address = 0x68    # Hexadecimal Address of Accelerometer

#PARAMETERS AND RECORDS IN HEXADECIMAL
      MPU6050

ACCEL_2G = 0x00 # Scales
ACCEL_4G = 0x08
ACCEL_8G = 0x10
ACCEL_16G = 0x18

SCALE_MULTIPLIER_2G = 4/65535 #Multiply for G
SCALE_MULTIPLIER_4G = 8/65535
SCALE_MULTIPLIER_8G = 16/65535
SCALE_MULTIPLIER_16G = 32/65535

BW_RATE_260HZ = 0x00
BW_RATE_184HZ = 0x01
BW_RATE_94HZ = 0x02
BW_RATE_44HZ = 0x03
BW_RATE_21HZ = 0x04
BW_RATE_10HZ = 0x05
BW_RATE_5HZ = 0x06

SCALE_REGISTER = 0x1C  #Controls scales
BW_REGISTER = 0x1A  #Controls bandwidth
MODE_REGISTER = 0x6B  #Controls measurement
      mode

MEASURE_MODE = 0x00

# —— Functions ——

def convert_LSBandMSB(raw_val):
  convert_val = (raw_val[0]<<8) + raw_val[1]
  if (convert_val>=0x8000):
     return -((65535 - convert_val) + 1) #MAIOR
       NUMERO PARA 16BYTES: 2^16
  else:
     return convert_val

# ON MPU6050 MSB COMES BEFORE LSB AND
      DATA IS IN 16BITS

def option_scale(scale):
  if scale == int(2):
     return [ACCEL_2G,SCALE_MULTIPLIER_2G]
  elif scale == int(4):
     return [ACCEL_4G,SCALE_MULTIPLIER_4G]
  elif scale == int(8):
     return [ACCEL_8G,SCALE_MULTIPLIER_8G]
  elif scale == int(16):
     return [ACCEL_16G,SCALE_MULTIPLIER_16G]


def option_banda(banda):
  if banda == int(5):
     return BW_RATE_5HZ
  elif banda == int(10):
     return BW_RATE_10HZ
  elif banda == int(21):
     return BW_RATE_21HZ
  elif banda == int(44):
     return BW_RATE_44HZ
  elif banda == int(94):
     return BW_RATE_94HZ
  elif banda == int(184):
     return BW_RATE_184HZ
  elif banda == int(260):
     return BW_RATE_260HZ

# USER INPUTS

RANGE_XG=int()
scale = int(input("Scale [2,4,8,16]:"))
RANGE_XG = option_scale(scale)

BANDW_XHZ=int()
banda = int(input("Bandwidth [5, 10, 21, 44, 94, 184,
      260]:"))
BANDW_XHZ = option_banda(banda)

freqInput = input("Measuring frequency[Hz]:")

#ene= input("2^(?):")  #nsample
#nsamp=2**int(ene)         #nsample
```

```
minutes = input(''Duração da Medição[m]:'')
seconds = input(''Duração da Medição[s]:'')
duration = timedelta(0,int(seconds),0,0,int(minutes))

time.sleep(1)

# MEASUREMENT CONFIGURATION
bus.write_byte_data(address, MODE_REGISTER,
    MEASURE_MODE)
alem=bus.read_byte_data(address, MODE_REGISTER)
if alem == MEASURE_MODE:
  print(''CORRECTLY CONFIGURED MODE'')
time.sleep(0.2)
bus.write_byte_data(address, BW_REGISTER,
    BANDW_XHZ)
allm=bus.read_byte_data(address, BW_REGISTER)
if allm == BANDW_XHZ:
  print(''BANDWIDTH CORRECTLY CONFIGURED'')
time.sleep(0.2)
bus.write_byte_data(address, SCALE_REGISTER,
    RANGE_XG[0])
alum=bus.read_byte_data(address, SCALE_REGISTER)
if alum == RANGE_XG[0]:
  print(''SCALE CORRECTLY CONFIGURED'')

print(str(RANGE_XG[1]))

time.sleep(1)

# TRANSFORM THE STRING FREQUENCY INPUT
    TO DATETIME
milsec = float(1000.0/float(freqInput))
if milsec==1000:
  period=timedelta(0,1)
elif milsec%1==0:
  period=timedelta(0,0,0,int(milsec))
else:
  period=timedelta(0,0,(milsec-int(milsec))*1000,int(
    milsec))

# Creating list where data will be saved with append
tempo=[]
accdata=[]
t=0
time.sleep(1)

N_freq=int(input('======== Enter number of
    frequencies to find:'))

while True:
init = input(''Start Data Acquisition (y/n):'')
 if init == ''y'':
   break
 else:
   pass

# LOOP DATA ACQUISITION

print('Start of Data Acquisition')

previousMillis = datetime.now()
start= datetime.now()
endloop=datetime.now()+duration

while datetime.now().time() <endloop.time():
 currentMillis = datetime.now()
 if (currentMillis - previousMillis >=period):
   previousMillis = datetime.now()
   accdata.append([[bus.read_byte_data(address,0x3B),
            bus.read_byte_data(address, 0x3C)],
           [bus.read_byte_data(address, 0x3D),
            bus.read_byte_data(address, 0x3E)],
           [bus.read_byte_data(address, 0x3F),
            bus.read_byte_data(address, 0x40)]])
   tempo.append(datetime.now())
 end=datetime.now()
 acqt = end-start

print('End of Data Acquisition')

# CONVERSION OF INPUTS IN LSB AND MSB FOR
    GRAVITY

for i in range(len(tempo)):
  for j in range(0,3):
    accdata[i][j] = convert_LSBandMSB(accdata[i][j]) *
      RANGE_XG[1]

# FILE OPENING
ttime=[]
opentime = datetime.now()
nome = opentime.strftime('MPU_data'+'(%d|%m|%Y_%H:
    %M: %S)')
saida = open(nome+''.csv'',''w'')

for i in range(len(tempo)):
 saida.write(str(i+1)+'';'') # MEASUREMENT
    NUMBER
 saida.write(str(tempo[i].time())+'';'') # TIME
 saida.write(str((tempo[i]-tempo[0]).total_seconds())+'';'')
 saida.write(str(accdata[i][0])+'';'') # Accelerometer X
 saida.write(str(accdata[i][1])+'';'') # Accelerometer y
 saida.write(str(accdata[i][2])+'';'') # Accelerometer z
 saida.write(str(acqt)+''\n'')
 ttime.append((tempo[i]-tempo[0]).total_seconds())
saida.close()

#Fourier Transform

acc=transpose(accdata)
sig=acc[2]
time_step=acqt.total_seconds()/len(sig)
Fs=1/time_step

from scipy import fftpack

sample_freq = fftpack.fftfreq(sig.size, d=time_step)
    # generates sampling frequencies
sig_fft = fftpack.fft(sig)              # calculates the fast
    Fourier transform
pidxs = where(sample_freq > 0)              # Only the
    positive part of the spectrum
will be used.
freqs = sample_freq[pidxs]
power = abs(sig_fft)[pidxs]

print (''|Fininish FFT'')

xcoords=zeros((N_freq))
DT=int(len(freqs)/N_freq)

for i in range(N_freq):
 fr=freqs[(i)*DT: DT*(i+1)]
 xcoords[i]=fr[argmax(power[(i)*DT: DT*(i+1)])]

# FILE NAMES TO BE SAVED IN SD

dataname=nome+''.csv''
specname=opentime.strftime('MPU_Specgram_'+'(%d|%m|
    %Y_%H: %M: %S)'+'.png')
figname = opentime.strftime('MPU_Figure_'+'(%d|%m|%
    Y_%H: %M: %S)'+'.png')

# GRAPH

import matplotlib.pyplot as plt

plt.figure(1,figsize=[12,6])
plt.style.use('classic')
```

```
plt.subplot(2, 1, 1)
plt.plot(ttime,sig,color='#2f4875',linewidth=0.8)
plt.xlabel('Time [s]')
plt.ylabel('Acceleration [g]')

plt.subplot(2, 1, 2)
plt.plot(freqs, power,color='#2f4875',linewidth=0.8)
plt.xlabel('Frequency [HZ]')
plt.ylabel('Power [dB]')
for xc in xcoords:
  plt.axvline(x=xc,
        linestyle='--',
        linewidth=1,
        color='#FF4040',
        label=str(round(xc,4))+'Hz')
plt.legend()
plt.grid(True)

plt.savefig(figname)
plt.show(block=False)
plt.pause(5)
plt.close()

# SPECTROGRAM:

plt.figure(2,figsize=[12,6])
plt.style.use('classic')

plt.subplot(2,1,1)
plt.plot(ttime,sig,color='#2f4875',linewidth=0.8)
plt.xlabel('Time [s]')
plt.ylabel('Acceleration [g]')
plt.subplot(2,1,2)
powerSpectrum,frequenciesFound,time,imageAxis=plt.
      specgram(sig,
                              Fs=Fs,
                              noverlap=255,
                              cmap='Spectral')
plt.xlabel('Time [s]')
plt.ylabel('Frequency [Hz]')
plt.savefig(specname)
plt.show(block=False)
plt.pause(5)
plt.close()

print ('| -------- Figure Saved -------- |')

# CLOUD STORAGE:

entrada=(figname,dataname,specname)

while True:
  resp=input('Save in dropbox (y/n):\t')

  if resp=='y':
    name_data=input('Enter data file name:\t')
    name_fig=input('Enter figure file name:\t')
    name_spec=input('Enter specgram file name: \t')
    saida = ('/E.R.A/'+name_fig+opentime.strftime('(%d
      |%m|%Y_%H: %M:
%S)')+'.png',
          '/E.R.A/'+name_data+opentime.strftime('(%d|%m
      |%Y_%H: %M:
%S)')+'.csv',
          '/E.R.A/'+name_spec+opentime.strftime('(%d|%m
      |%Y_%H: %M:
%S)')+'.png')

    print('| ----- Start file writing in Dropbox ----- |')

    import dropbox

    class TransferData:
      def __init__(self, access_token):
        self.access_token = access_token

      def upload_file(self, file_from, file_to):
        """upload a file to Dropbox using API v2
        """
        dbx = dropbox.Dropbox(self.access_token)

        with open(file_from, 'rb') as f:
          dbx.files_upload(f.read(), file_to)

    def main():
      access_token = 'PUT THE TOKEN OF THE
       ACCOUNT YOU WANT TO
SAVE HERE'
      transferData = TransferData(access_token)

      for i in range(3):
        file_from = entrada[i]
        file_to = saida[i]
        # API v2
        transferData.upload_file(file_from, file_to)

    if __name__ == '__main__':
      main()
    break
  elif resp=='n':

    break
  else:
    pass

print ('| ----- Completely Finish ----- |')
```

References

[1] P. Avitabile, *Experimental modal analysis—A simple non-mathematical presentation,* available in: https://www.uml.edu/docs/s-v-Jan2001_Modal_Analysis_tcm18-189939.pdf.

[2] M.L. Chandravanshi and A.K. Mukhopadhyay, in *Proceedings of the ASME 2013 International Mechanical Engineering Congress and Exposition* (San Diego, 2013).

[3] W. He and J. Liu, *Active Vibration Control and Stability Analysis of Flexible Beam Systems* (Springer, Beijing, 2019).

[4] C. Ratcliffe, D. Heider, R. Crane, C. Krauthauser, M.K. Yoon and J.W. Gillespie, Composite Structures **82**, 61 (2008).

[5] S.B. Chaudhury, M. Sengupta and K. Mukherjee, International Journal of Scientific Engineering and Research (IJSER) **2**, 3 (2014).

[6] D.M. Lima, P.A. López-Yánez and M.A Pereira, Lat. Am. j. solids struct. **16**, 15 (2019).

[7] P.S. De Brito and Humberto Varum, *Accelerometers: Principles, Structure and Applications* (Nova Science Pub, New York, 2013).

[8] D.K. Fisher and P.J. Gould, Modern Instrumentation **1**, 8 (2012).

[9] A. D'Alessandro, R. D'Anna, L. Greco, G. Passafiume, S. Scudero, S. Speciale and G. Vitale, in *IEEE International Symposium on Inertial Sensors and Systems—INERTIAL* (Moltrasio, 2018).

[10] Y. Xiuping, L. Jia-Nan and F. Zuhua, in *8th International Conference on Intelligent Computation Technology and Automation—ICICTA* (Nanchang, 2015).

[11] L. Liu, D. Zheng, X. Liu, Chinese Journal of Medical Instrumentation **39**, 330 (2015).

[12] S. Yan-Ping, G. Mao-fa, A. Bin, Z. Jian-yu and Z. Xu-jie, Journal of Measurement Science & Instrumentation **5**, 19 (2014).

[13] Y.-g. Gu, Y.-j. Shi, X.-j. Zhou and C.-h. Shi, Manufacturing Automation **32**, 15 (2010).

[14] B.V.S. Krishna, J. Oviya, S. Gowri and M.A. Varshini, *in Second International Conference on Science Technology Engineering and Management—ICONSTEM* (Chennai, 2016).

[15] R. Singh, A. Gehlot, L.R. Gupta, B. Singh and M. Swain, *Internet of Things with Raspberry Pi and Arduino* (CRC Press, London, 2019).

[16] S. Ambre, M. Masurekar, S. Gaikwad, in *Modern Approaches in Machine Learning and Cognitive Science: A Walkthrough. Studies in Computational Intelligence* edited by V. Gunjan, J. Zurada, B. Raman and G. Gangadharan (Springer, Cham, 2020), p. 1, v.885.

[17] M. Varanis, A.L. Silva, P.H. Brunetto and R.F. Gregolin, *Rev. Bras. Ensino Fís.* **38**, 1301 (2016).

[18] M. Varanis, A.L. Silva, A.G. Mereles, Rev. Bras. Ensino Fís. **40**, e1304 (2017).

[19] M. Varanis, A. Silva, A. Mereles and R. Pederiva, J Braz. Soc. Mech. Sci. Eng. **40,** 527 (2018).

[20] T.T Duc, T.L Anh and H.U. Dinh, in *International Conference on Advances in Computational Mechanics* (Springer, Singapore, 2017).

[21] J.M. Mahoney and R. Nathan, in *ASEE Annual Conference and Exposition, Conference Proceedings* (Columbus, 2017).

[22] C. Reddy, S. Shenoy, R. Sharma and S. Ramesh, Vibroengineering PROCEDIA **29**, 1 (2019).

[23] A. Devices, *Adxl345 datasheet*, available in: https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf

[24] https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf, accessed in 04/03/2020.

[25] https://www.nxp.com/docs/en/data-sheet/MMA7455L.pdf, accessed in 04/03/2020.