2. Project Accelerometer: Obtaining data from an accelerometer for real-time NN evaluation

> ***This page aims to explain the process of obtaining, processing and using vibration signals generated by the ICP® Accelerometer.***

Introduction

This experiment is part of a broader research initiative aimed at developing and implementing machine learning algorithms to detect anomalous behavior in multirotors. The primary objective is to identify potential faults before they escalate into critical failures.

To achieve this, we initially trained a **Convolutional Neural Network (CNN)** with an accuracy exceeding **85%**. The next step involves obtaining real-world data using an **ICP® Accelerometer**. This data will be utilized for model retraining and evaluation, ensuring not only the accuracy of the model but also the suitability and reliability of the collected data.

The following tasks outline the experimental procedure:

1. Configure and connect an **Arduino Mega 2560 PRO** to the computer.
2. Develop a basic program in **Arduino IDE** to collect data from the accelerometer.
3. Connect the accelerometer and verify its proper operation.
4. Collect real-time vibration data from a **multirotor** for further analysis.
5. Show and study the data recolleted

Inventory (materials)

To aim this taks, we count on the following items:

PC Dell XPS 15 7590

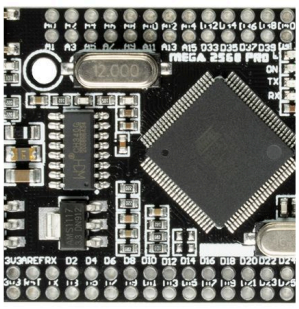
This laptop serves as the main development and processing unit, running the Arduino IDE, logging data, and performing signal analysis. It counts with:

- Ubuntu 22.04.5 LTS
- Intel® Core™ i7-9750H CPU @ 2.60GHz × 12
- NVIDIA TU117M [GeForce GTX 1650 Mobile / Max-Q] / Mesa Intel® UHD Graphics 630 (CFL GT2)

Arduino Mega 2560 PRO

The Arduino board is responsible for interfacing with the accelerometer, collecting analog data, and transmitting it to the PC. It provides multiple input channels, sufficient memory, and processing power to handle sensor data in real-time.

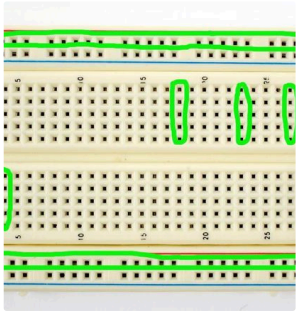
💡 **Arduino Mega 2650 PRO Datasheet:**
https://www.enmindustry.de/WebRoot/Store31/Shops/88169453/5FFE/0DC7/1617/A559/78B1/0A0C/6D12/6D9F/Mega2650PRO-Datasheet.pdf

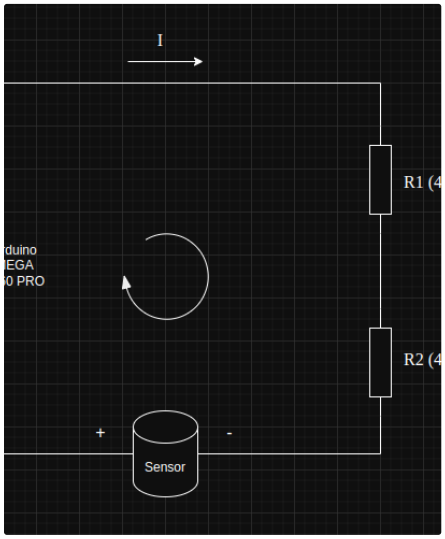


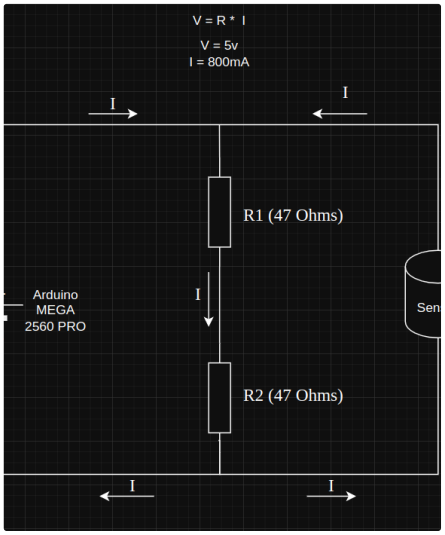
Test borad

The test board allows for an organized connection of the circuit components. It may be usefull to use a paralell circuit with two ressistance to dispel some power, if needed. Otherwise, a simple in-line circuit will be done

Proper wiring will also be required to ensure the connections between the accelerometer, signal conditioner, and Arduino.







ICP Accelerometer - Model 352A92

This high-precision accelerometer is used to measure vibration signals from the multirotor.

💡 **Accelerometer Datasheet:** Model 352A92 | PCB Piezotronics

Four-Channel, ICP Sensor Signal Conditioner - Model 482C15

The signal conditioner ensures that the accelerometer's output is properly amplified and filtered before being processed by the Arduino:

1. **Constant Current Supply**: The signal conditioner provides a constant current (typically between 2 and 20 mA) necessary to power the ICP® accelerometer.
2. **Signal Conversion**: It converts the high-impedance signal from the accelerometer into a low-impedance signal, suitable for reading by devices like the Arduino.
3. **Filtering and Amplification**: It may include filtering and amplification functions to improve signal quality and eliminate noise.

💡 **Sensor conditioner Datasheet:** Model 482C15 | PCB Piezotronics

Development of the experiment

Configure Arduino

Configure and connect an Arduino Mega 2560 PRO to the computer:

- First we install the **Arduino IDE** and necessary drivers on the PC. Software
- Connect the **Arduino Mega 2560 PRO** to the computer via USB cable.
- Check the **Serial Port** and confirm that the Arduino is detected.
- Because of the Serial Ports were not detected, these commands helped solve the
- issue after retrying: `sudo usermod -a -G dialout $USER` , `sudo apt remove brltty` .
- Then this command, to check if the Serial Port is detected: `ls /dev/ttyUSB*`
- Finally upload a basic test script to verify communication between the PC and the Arduino board.

```
void setup() {
  // Initialize the onboard LED pin as an output
  pinMode(LED_BUILTIN, OUTPUT);
  // Start the Serial communication at 9600 baud rate
  Serial.begin(9600);
  // Print a message to the Serial Monitor
  Serial.println("Arduino Mega is active and well detected!");
}

void loop() {
  // Turn the LED on (HIGH is the voltage level)
  digitalWrite(LED_BUILTIN, HIGH);
  // Wait for a second
  delay(1000);
  // Turn the LED off by making the voltage LOW
  digitalWrite(LED_BUILTIN, LOW);
  // Wait for a second
  delay(1000);
}
```

Implement programm

Use the Arduino IDE to develop a basic programm that:

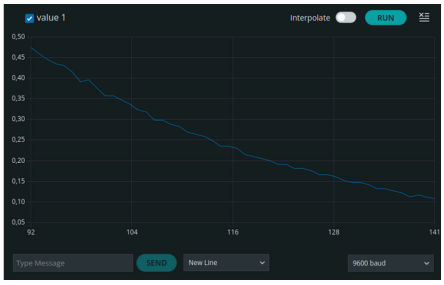
- Reads **analog signals** from the accelerometer through the **A0** pin. Previously, triend in one of th eDX pins but did not work becaouse we are reaching for an analog connection, thus we use one of the analog AX pins.
- Implements **ADC (Analog-to-Digital Conversion)** to translate sensor outputs into meaningful voltage readings ranging from **0-5V**.
- Dinamically prints the data in the Serial Monitor.
- Optionally stores and format sensor values in an appropriate structure (CSV a first) for further analysis.

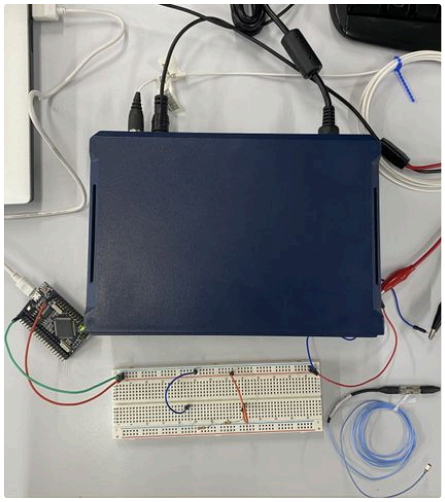
```
#define PIN_VIBRATION A0

void setup() {
    Serial.begin(9600);
    pinMode(PIN_VIBRATION, INPUT);
}

void loop() {
    int vibration = analogRead(PIN_VIBRATION);
    float voltage = (vibration * 5.0) / 1023.0;

    Serial.println(voltage, 3);

    delay(500);
}

```

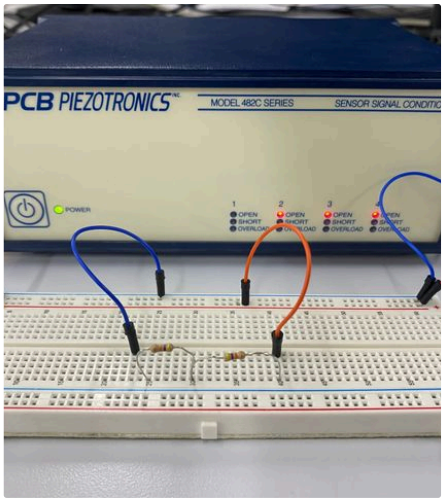
Test accelerometer

Connect the accelerometer and test its correct operation:

- Wire the **ICP® Accelerometer** to the **Four-Channel Signal Conditioner** ensuring correct pin configuration.
- Supply the **required excitation voltage** of **20-30V DC** to the signal conditioner.
- Check the output signal using the **Arduino Serial Monitor** and the **Arduino Serial Plotter** to confirm data consistency.
- **After checking, we do not see any fluctuation on the vibraations after a first acummulated frecuency (see ilustration)**

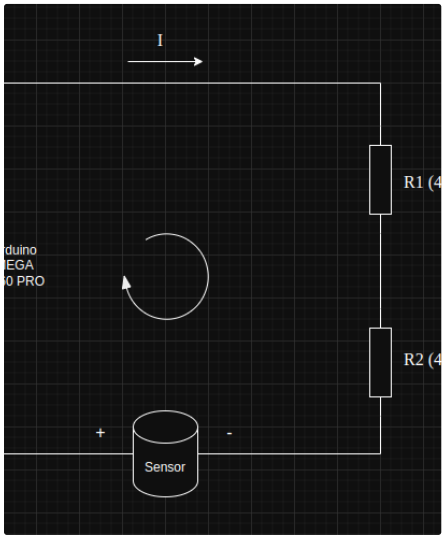




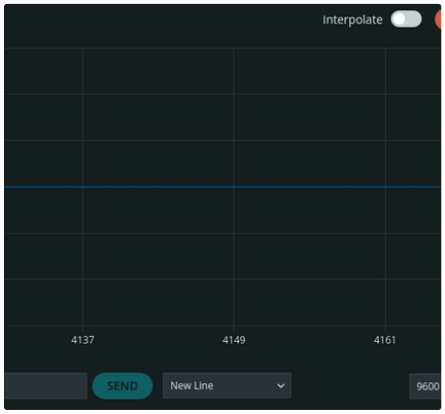


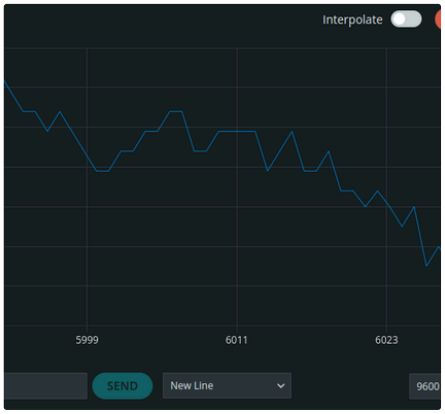
After a second try, we finally managed to correctly detect vibrations thanks to these modifications:

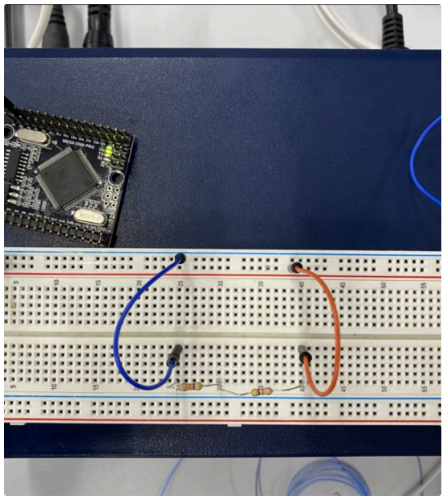
- The arduino code was modified by decreasing the **delay** by 90% (from 500 to 50) and increasing the **resolution** of the Analog-to-Digital Converter (ADC) from default (5V) to 1.1V.
- The circuit was rearanged to the first option showed in the Inventory, due to the lack of efectivity of the resistances.
- Finally, to test the accelerommeter with more accurate data, we make use of a smartphone vibrator motor to tests the data recollected.

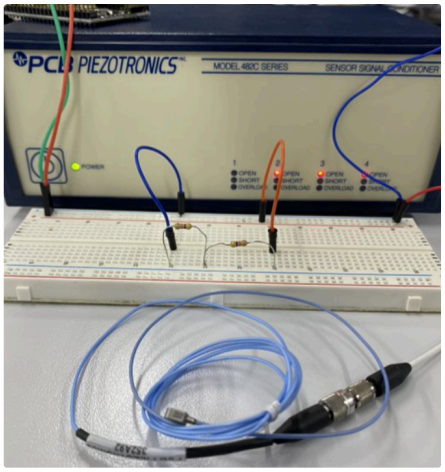



```
1
2  #define PIN_VIBRATION A0
3
4  void setup() {
5      Serial.begin(9600);
6      analogReference(INTERNAL1V1);
7      pinMode(PIN_VIBRATION, INPUT);
8  }
9
10 void loop() {
11     int vibration = analogRead(PIN_VIBRATION);
12     float voltage = (vibration * 5.0) / 1023.0;
13
14     Serial.println(voltage, 3);
15
16     delay(50);
17 }
```









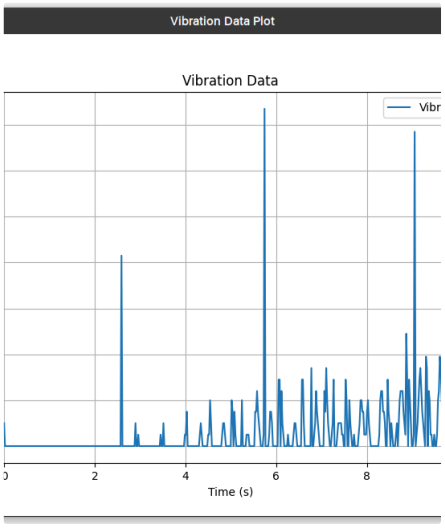
Obtain data

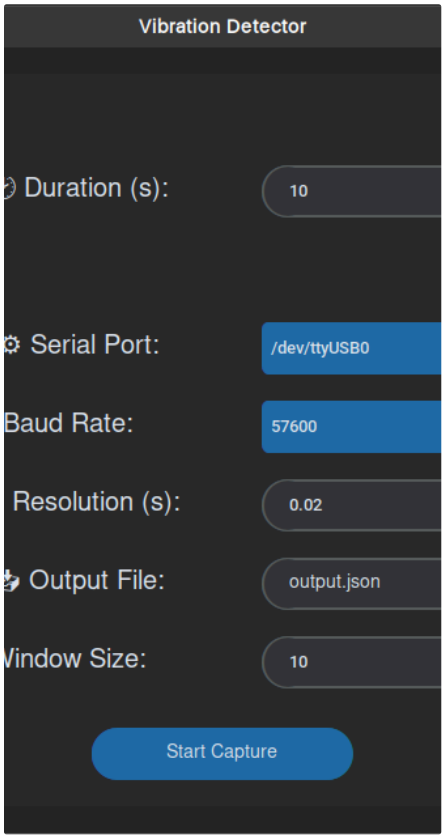
After the successful connection and data detection of the accelerometer using the Arduino IDE, the implementation of a Python-based interface was carried out. This interface will allow to configure various parameters for **capturing accelerometer signals**. Additionally, the program is responsible for **storing the data, displaying the windowed data, and plotting the captured vibrations** in real-time.

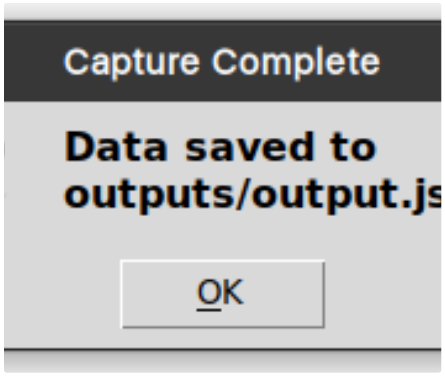
The program requirements are as follows:

- The program will allow users to select the interface to which the Arduino is connected.
- It will alert the user if no device is detected on the selected interface.
- Users will be able to configure the capture duration (seconds), time window size (number of windowed elements), and resolution (seconds).
- It will display and store a time-series graph showing the captured vibrations.
- Data will be stored in JSON format, with each vibration entry accompanied by a timestamp.
- Users can specify the output file names.
- The program will allow configuration of the Arduino's data read/write speed (baud rate), restricting selection to valid values only.
- A brief waiting period will be implemented before starting data capture.
- Users can cancel the operation at any time to prevent errors.
- Appropriate constraints and validations will be in place to prevent incorrect values from being entered in fields such as duration and resolution.

This program enables real-time and controlled capture of accelerometer-detected vibrations. Furthermore, it allows data storage and visualization while offering parametrization and configurability for conducting necessary tests. Below, the interface and results obtained from a 10-second capture are presented.







…

Window 29: [(5.761467456817627, 0.0), (5.781881332397461, 0.0), (5.8023681640625, 0.0), (5.823009490966797, 0.0), (5.843634128570557, 0.005), (5.864079236984253, 0.015), (5.8845953941345215, 0.015), (5.905025482177734, 0.01), (5.925647497177124, 0.0), (5.946094751358032, 0.0)]
Window 30: [(5.9667158126831055, 0.0), (5.9875757694244385, 0.0), (6.007988452911377, 0.0), (6.03075098991394, 0.0), (6.05114221572876, 0.029), (6.0735039710998535, 0.029), (6.093979120254517, 0.0), (6.114619493484497, 0.024), (6.135035991668701, 0.01), (6.155510663986206, 0.005)]

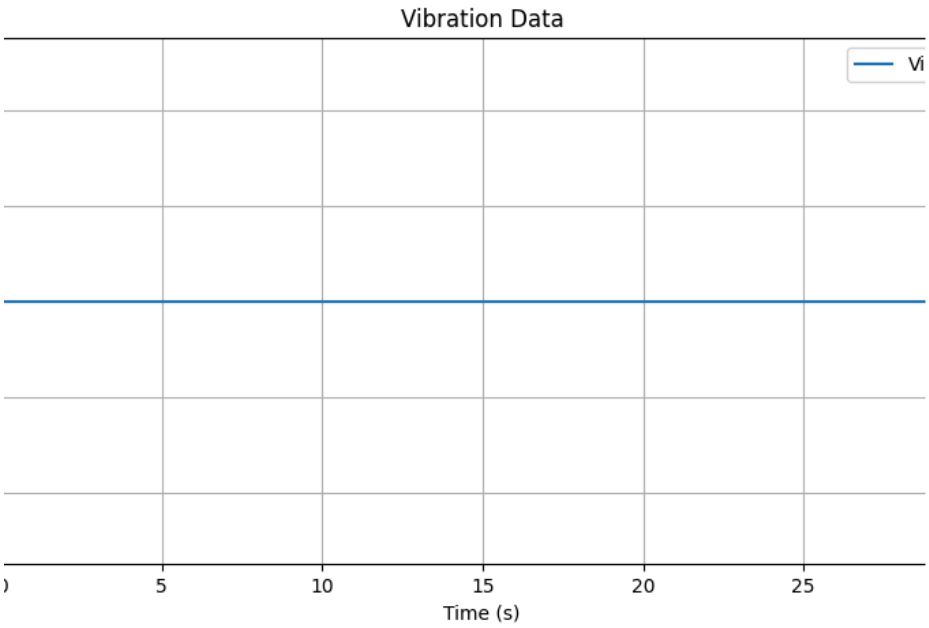
…

Show captured data

Finally, using the developed program, various tests will be conducted to evaluate:

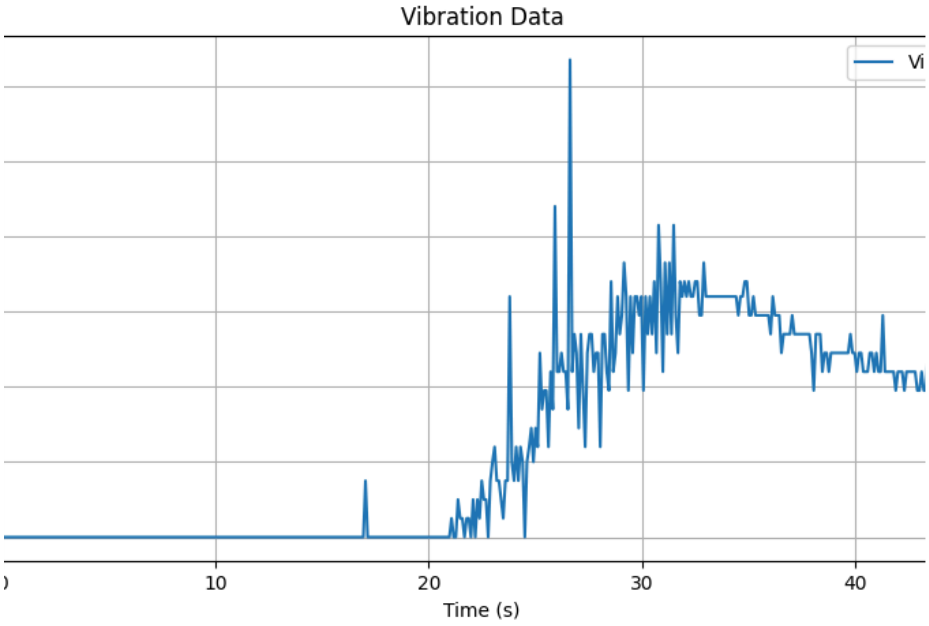
- The correct detection of data by the accelerometer.
- The maximum and minimum resolution achievable.
- The influence of the board's read/write speed and its optimal configuration.

Firstly, we want to verify that the accelerometer can correctly detect vibrations. To do this, we conducted a test in which we did not touch the accelerometer to determine whether it detects noise that could render the data capture unusable.

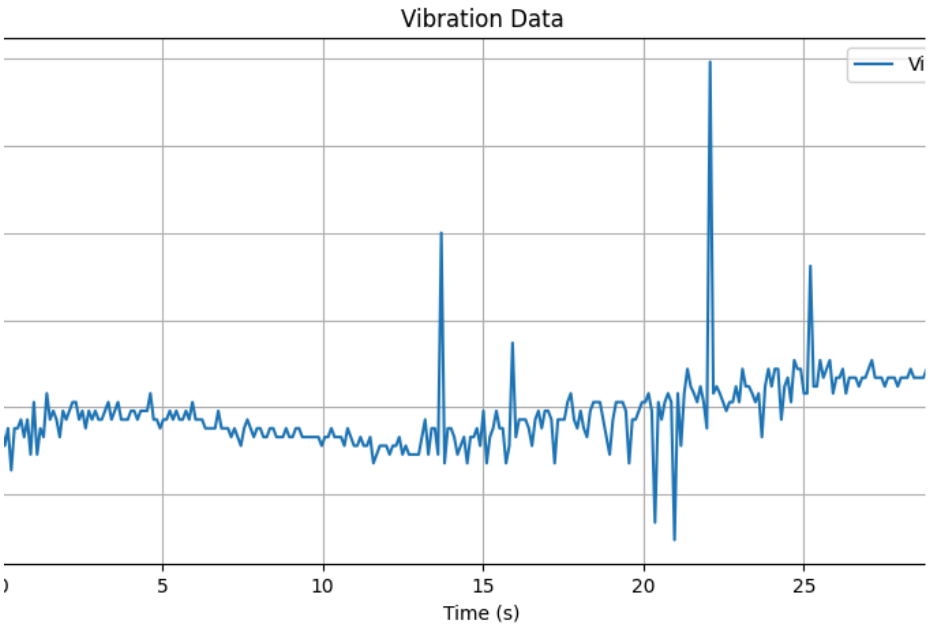


As seen in the graph, this is not the case, as no data is recorded when the accelerometer remains stationary.

Next, we held the accelerometer still and then shook it vigorously. The resulting graph clearly distinguishes between these two intervals.



Finally, we performed a last test in which we shook the accelerometer with varying intensities, confirming that the program can detect and capture significant variations in vibration intensity.

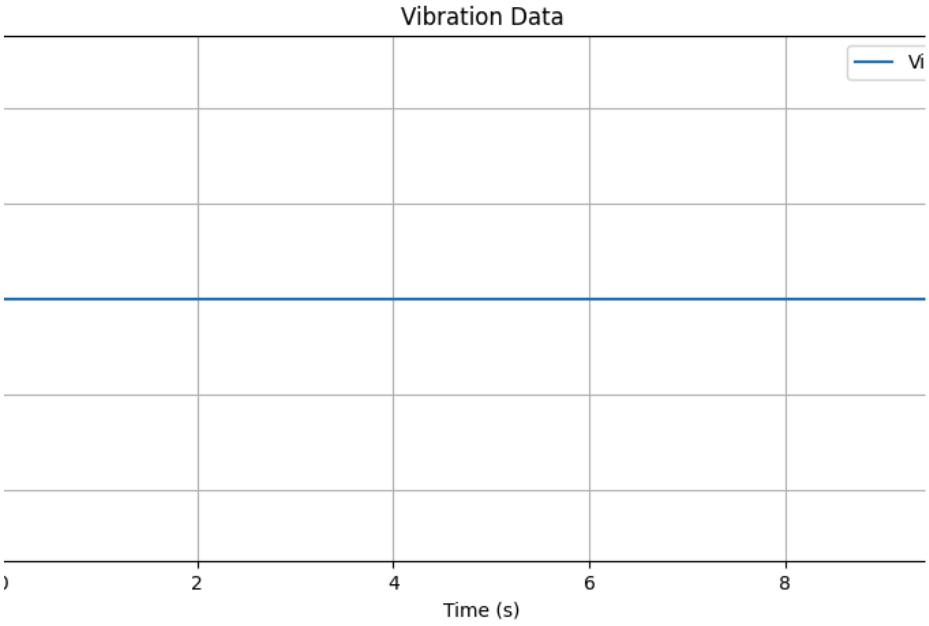


Following this initial analysis, we proceeded to study the different factors that may influence the sensitivity of data capture. The two primary parameters are resolution and the Arduino's data read speed.

To determine the maximum and minimum **resolution** at which we can obtain reliable data, we conducted multiple tests by adjusting the "Resolution" parameter. This parameter, measured in seconds, is inversely proportional to frequency: the lower the time (in seconds), the higher the frequency. Our goal is to achieve the highest possible capture frequency to ensure the most accurate vibration detection. The following images display 10-second captures performed at 9600 bps using different resolutions.

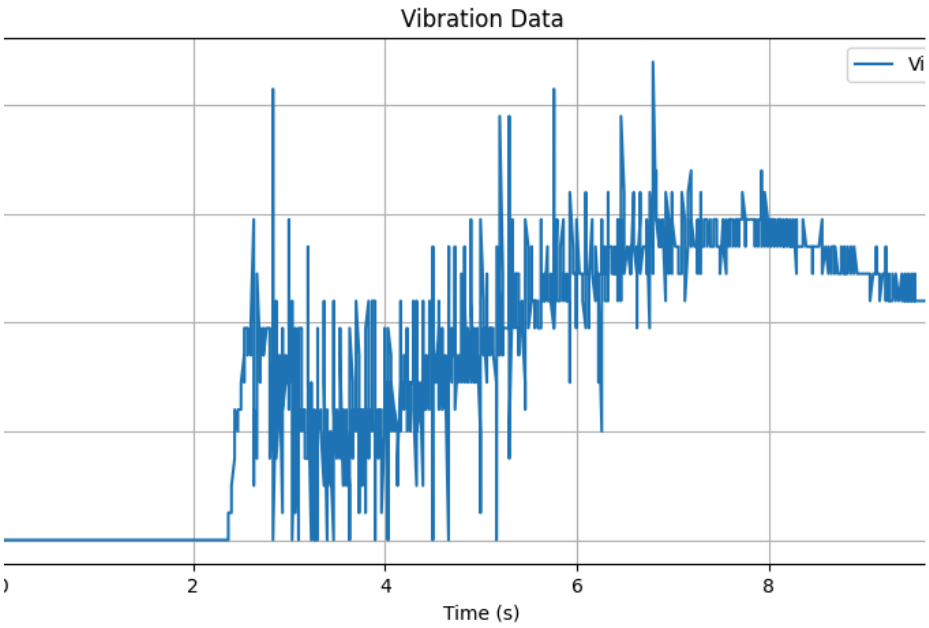
From the first image, we conclude that using resolutions with delays greater than 700ms leads to a loss of vibration detection. The **average sampling frequency** is approximately **1.43 Hz**, meaning that not even 2 samples are being captured per second.

Note that other experiments have been done where some vibrations are detected, but were too low and so few to be considered conclussive.

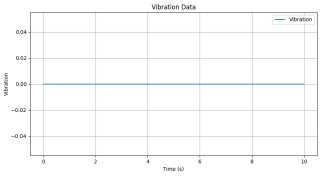
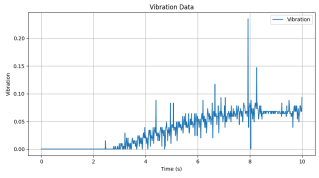
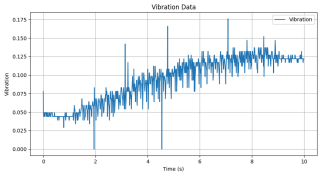


Conversely, using the lowest possible resolution (0.000s delay, which the arduino detects as the minimum **0.00016**s) yields significantly more sensitive results, producing a more detailed graph.

The **average sampling frequency** is approximately **156 Hz**, meaning that around 156 samples are being captured per second.



Additionally, experiments were conducted using different **data read speeds** (baud rate) on the Arduino board, beyond the default value of 9600 bps. All tests were performed using a resolution of 0.01s over 10-second captures.

		
Baud rate = **1200** / Frequency = **49** Hz / Resolution = 0.01 s	Baud rate = **9600** (default) / Frequency = **94** Hz / Resolution = 0.01 s	Baud rate = 57600 / Frequency = **93.73** Hz / Resolution = 0.01 s

We observed that higher read speeds result in a more detailed captured data, allowing for an improved data capture analysis. In any case, frequency is not even more precise at higher bad rate levels.

Study and improove captured data

After analysing the data regarding requency sensibiity, the following feedback was given:

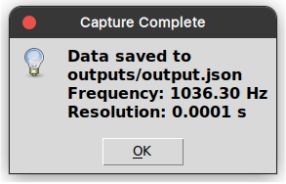
1. Maintaining a fixed data reading frequency is not meaningful in this context. All incoming data from the serial port should be read as it arrives, rather than sampling at a predefined resolution. Ultimately, the data is being generated at a frequency controlled by the Arduino. Introducing an additional limitation by downsampling this frequency does not seem justified.
2. The current code follows a loop workflow that reads from the analog input, sends data through the serial port and pauses for *T* seconds. This results in a data generation frequency of *T* + *execution time*, which is imprecise. This may explain why the reading frequency does not reach 100 Hz in the documented script, stabilizing around 94 Hz.

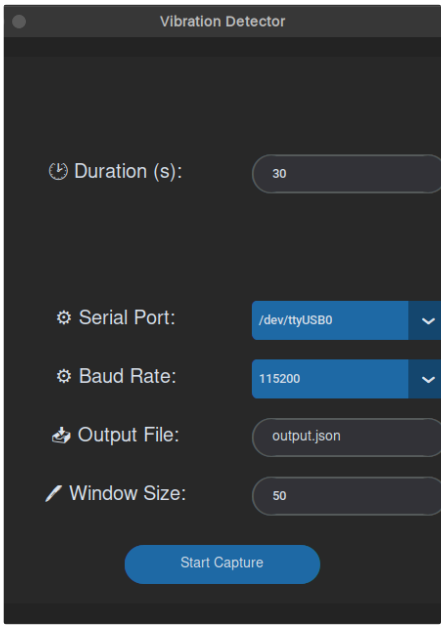
To address both issues, the arduino code has been change to implement a timer and starting with an initial value of **100 Hz and to also consider time outputs directly given by the Arduino**. Also, the Python code has been change to directly detect the data read by the serial port instead of stablishing a resolution parameter.

```ino
#include <avr/io.h>
#include <avr/interrupt.h>

#define PIN_VIBRATION A0
#define SAMPLE_FREQUENCY 1000  // Sample freq (Hz)

void readVibration() {
    int vibration = analogRead(PIN_VIBRATION);
    float voltage = (vibration * 5.0) / 1023.0;
    unsigned long timeStamp = micros();

    // Output
    Serial.print(timeStamp);
    Serial.print(", ");
    Serial.println(voltage, 3);
}

void setupTimer() {
    cli(); // Disable global interrupts

    TCCR1A = 0; // Configure Timer1 in CTC mode
    TCCR1B = 0;
    TCCR1B |= (1 << WGM12); // Set CTC mode
    TCCR1B |= (1 << CS11);  // Prescaler 8

    OCR1A = (16000000 / (8 * SAMPLE_FREQUENCY)) - 1; // Compare match value for 100 Hz
    TIMSK1 |= (1 << OCIE1A); // Enable compare match interrupt

    sei(); // Enable global interrupts
}

ISR(TIMER1_COMPA_vect) {
    readVibration();
}

void setup() {
    Serial.begin(115200);
    analogReference(INTERNAL1V1);
    pinMode(PIN_VIBRATION, INPUT);

    setupTimer();
}

void loop() {

}
```

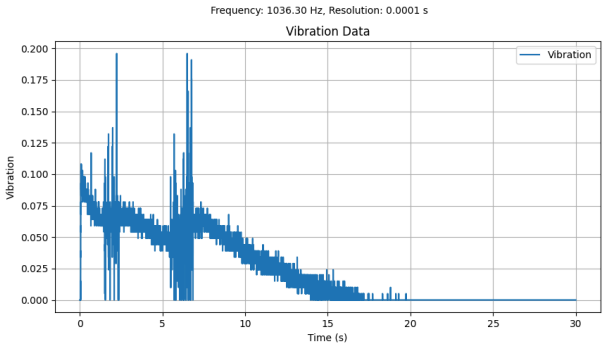
After severals modification on the Python code, we have obtained a full dataset with a desire maximun frequency. The following test shows how changing the values on the Arduino IDE as well as the default values on the Python app we finally manage to obtain max detailed outputs:

```python
# PYTHON APP DEFAULT VALUES

DURATION_DEFAULT = "30"
SERIAL_PORT_DEFAULT = "/dev/ttyUSB0"
OUTPUT_FILE_DEFAULT = "output.json"
WINDOW_SIZE_DEFAULT = "50"
BAUD_RATE_DEFAULT = "115200"
```





Regarding the plotted data, in this final example we have produce a more intense movement at the beginning followed by a progressive stop. We can appreciate that the requency as well as the resolution show the expected value, and the wave function responds as expected.



Final experimentation

Finally, after developing this study about how the accelerometer work and how we process the data, we will now proceed to put to accelerometer data capture to its limit:

1. **ADC Voltage Scale & Resolution**

   We will test the different voltage values of the Arduino Mega and determine which one is the apropriate one for the real use case. We will test the following reference values always at frequency=1000Hz, 5s captures and `115200` baud rate:

- `analogReference(INTERNAL1V1)`
- `analogReference(INTERNAL2V56)`
- `analogReference(DEFAULT)`

Conclusion:

- Our experiments confirmed that when using `analogReference(INTERNAL1V1)`, the ADC operates within an approximately 1.1V range. This setting results in lower and step-like voltage readings due to the limited resolution of the 10-bit ADC (0–1023 steps). With this analog reference we obtained an expected frequency of 997.0Hz.
- Trying other resolutions we still kept topped at around 900Hz frequencies not showing any noticeable difference.

---

2. **Sampling Frequency**

   Configure the sampling frequency to 10 Hz, 100 Hz, 500 Hz, 1000 Hz, and Maximum Admissible and compare its results. Each of the takes shown in the table have been repeated thrice for better presition, and the middle value has been taken.

- *Conclusion:*
  For each of the five sampling frequencies tested (10 Hz, 100 Hz, 500 Hz, 1000 Hz, and the maximum admissible rate), data was captured reliably over a 5-second period. The generated JSON files included vibration readings that closely matched the configured (expected) frequency, with the average measured frequency showing only minimal deviation. The calculated metrics—including mean, maximum, and minimum time differences between samples—remained within acceptable error margins. **The maximum reached frequency was reached at around 1KHz**. Finally, **the arduino stops showing significant data at around 15Hz, none at 10Hz.**

Expected freq (Hz)	Mean freq (Hz)	ΔT min	ΔT max	ΔT mean	Time source (pc/arduino)
1.00	**1.00**	`0.9999560000`	`1.0000039999`	`0.9999912000`	**pc** & **arduino**
10.00	**10.00**	`0.0999200000`	`0.0999720000`	`0.0999672542`	**pc** & **arduino**
100.00	**121.71**	`0.0001108646`	`0.0108549594`	`0.0082103245`	**arduino**
100.00	121.80	`0.0001100000`	`0.0108539999`	`0.0082100000`	**pc**
500.00	**500.05**	`0.0001122951`	`0.0108220577`	`0.0016419450`	**arduino**
500.00	609.01	`0.0001110081`	`0.0108602310`	`0.0082200001`	**pc**
1,000.00	755.06	`0.0000950000`	`0.0078000000`	`0.0013200000`	**arduino**
1,000.00	**907.22**	`0.0001279999`	`0.0020120000`	`0.0013235148`	**pc**
16,000.00	907.09	`0.0000900000`	`0.0075000000`	`0.0012000000`	**arduino**
16,000.00	**998.16**	`0.0000850000`	`0.0070001001`	`0.0011002100`	**pc**

After a fila arduino code refinement, we acheieve nearly perfect presition using the arduino timer, and similar results using the pc (Python) timer.

Expected freq (Hz)	Mean freq (Hz)	ΔT min	ΔT max	ΔT mean	Time source (pc/arduino)	
1.00	**4.682**	0.213	0.215	0.214	**arduino**	
1.00	5.764	0.001	0.214	0.173	**pc**	
10.00	**10.000**	0.098	0.100	0.099	**arduino**	
10.00	12.548	0.001	0.100	0.080	**pc**	
100.00	**100.000**	0.009	0.011	0.010	**arduino**	
100.00	127.165	0.001	0.011	0.008	**pc**	
500.00	**500.0000**	0.0009	0.0030	0.0020	**arduino**	
500.00	636.9257	0.00006	0.00803	0.00157	**pc**	
1,000.00	**1000.00000**	0.00001	0.00200	0.00100	**arduino**	
1,000.00	1272.52875	0.00006	0.00516	0.00078	**pc**	
1,500.00	3666.66666	0.00020	0.00100	0.00001	**arduino**	
1,500.00	**1766.49390**	0.00006	0.00349	0.00056	**pc**	

Observations from this new experiment:

- The **Arduino** maintains a **frequency very close to the expected** value at all levels (1 Hz, 10 Hz, 100 Hz, etc.), demonstrating stable timing.
- The **PC**, however, consistently measures **higher-than-expected frequencies**, especially at lower expected frequencies (e.g., 1 Hz expected → 5.76 Hz measured). This suggests that the PC introduces some form of sampling bias, possibly due to OS-level scheduling or variable loop execution times.
- The **Arduino** shows **small and stable variations** in ΔT, staying close to the theoretical values. At higher frequencies (e.g., 1000 Hz), it maintains a precise timing with **minimal jitter** (ΔT min: 0.00001 s, ΔT max: 0.002 s).
- The **PC** shows **much higher variability** in ΔT, especially at lower frequencies. The **ΔT min is sometimes extremely low (0.00006 s), while the ΔT max is significantly larger** than expected. This suggests the PC is **not executing measurements at perfectly uniform intervals**—likely due to OS multitasking, background processes, or limitations in software-based timing.

For achieving these results, the code as been finally modified as shown:

```
#include <avr/io.h>
#include <avr/interrupt.h>

#define SAMPLE_FREQUENCY 500 // Frequency in Hz

void setup() {
    Serial.begin(115200); // Initialize serial communication

    cli(); // Disable interrupts

    // Configure Timer1
    TCCR1A = 0; // Normal operation mode
    TCCR1B = (1 << WGM12) | (1 << CS11) | (1 << CS10); // CTC mode, Prescaler 64
    OCR1A = ((long)F_CPU / (64 * (long)SAMPLE_FREQUENCY)) - 1; // Corrected formula
    TIMSK1 = (1 << OCIE1A); // Enable Timer1 compare interrupt

    sei(); // Enable interrupts
}

ISR(TIMER1_COMPA_vect) {
    int sensorValue = analogRead(A0);
    unsigned long timestamp = millis();
    Serial.print(timestamp);
    Serial.print(",");
    Serial.println(sensorValue);
}

void loop() {
}

```

All the source data use to generate in both tables has been recollected and saved in the project's repository downloads y the `/outputs` file. Specifically, all the data named `1_output_xHz.json` is the one that has been used for the first experiments, and `2_output_xHz.json` for the latter.

Conclusions

The following key points summarize the findings of this study:

- **Accurate Vibration Detection:** The accelerometer demonstrated reliable performance in capturing vibration signals. No spurious noise was detected when the sensor remained stationary, and clear, distinct patterns were observed during shaking. This suggests that the system is capable of distinguishing between periods of no motion and vibrations.
- **Resolution Impact on Sampling Frequency:** The experiments highlighted that a high delay (around 700 ms) resulted in an average sampling frequency of approximately 1.43 Hz, which was inadequate for detailed vibration analysis. However, by reducing the delay to the minimum detectable limit of the sensor (around 0.00016 s), the sampling frequency increased but not as hightly as expected.
- **Influence of Baud Rate:** Further experiments with varying baud rates (1200, 9600, and 57600 bps) revealed that higher baud rates lead to more detailed data capture, although they did not significantly impact the frequency of data acquisition. This indicates that while a higher baud rate improves the resolution of captured data, it does not alter the fundamental sampling rate beyond the default setting.
- **Improvement in Data Analysis**
  - The experiments conducted on the ADC voltage scale and resolution confirmed that using `analogReference(INTERNAL1V1)` **provides a stable 1.1V**. Regardless of the reference voltage used, the measured frequency remained close to 900Hz / 1000 Hz, with no significant improvements observed.
  - Regarding sampling frequency, tests were performed at 10Hz, 100Hz, 500Hz, 1000Hz, and the maximum admissible rate. The Arduino **reliably captured data up to approximately 1kHz, with minimal deviations in expected frequency**. However, data collection **became inconsistent below 1,5KHz, with no reliable readings**. The collected data, stored in the project repository, validates these findings and serves as a reference for future optimizations. fadacatec-ondemand/predictive_maintenance_multirrotors