Instituto Superior Técnico

Big-Data Measuring Systems

Digital Acquisition and Processing & Cloud Sensor Platforms

Formal report

96261 – Jun Zhou

96523 – Duarte Miguel de Aguiar Pinto e Morais Marques

97381 – João Mendes das Neves Martins

Group 2

Prof. Luís Filipe Soldado Granadeiro Rosado

January 20th 2023

I. Introduction [1, 2, 3]

Digital conversion and sampling enable the information on analog signals to cross-over into the digital domain. An Analog to Digital Converter (ADC) is responsible for generating binary representations of the converted signals that are discrete in time and amplitude. Once in the digital domain, algorithms and communications are usually employed. Digital processing is particularly relevant concerning physical phenomena, since a considerable dimension reduction from the acquired signal can be achieved. The Nyquist-Shannon sampling theorem defines the time discretization conditions needed in order to capture all the required information on the resulting digital signal. More specifically, the theorem states that if a signal with limited bandwidth is sampled at twice its maximum frequency, it can be recuperated - otherwise, aliasing will occur.

Cloud sensor platforms have been particularly used to provide turnkey functionality in a Software as a Service (SaaS) model. This becomes increasingly more significant as M2M and IoT applications lead to a sharply growing amount of generated data. These platforms include backend resources from provisioning services, authentication services and databases to communication endpoints. Frontend applications are allowing to build customizable monitoring dashboards and execute data analytics. Mostly because of the easy integration of MATLAB numerical computing, the ThingSpeak platform is one of the most popular for scientific experiments. It may also be used to analyse signals from industrial motors and trigger eventually necessary maintenance action.

Unexpected equipment failures can be expensive and catastrophic, resulting in production downtime, replacement of parts and safety concerns. Vibration-based condition monitoring can be used to detect and diagnose machine faults and form the basis of a Predictive Maintenance strategy. Accelerometers are usually employed to transduce acceleration into an electrical signal later digitalized using a microcontroller. Carefully designed algorithms process the acquired signals and transform them into features for transmission through wireless connection.

This laboratory assignment initially demonstrates the spectrum of acquired signals with different characteristics and proves the aforementioned Nyquist-Shannon sampling theorem. Then, the experimental setup is used to acquire acceleration data when simulating different motor unbalancing situations and using different supply voltage values. Moreover, the respective MATLAB code is then connected with the cloud sensor platform ThingSpeak. Besides the integration, it is also demonstrated how cloud computing can be used to perform post-processing of sensors data and generate alarmistic.

II. Experimental setup

In this laboratory activity, the experimental setup shown in Figure 1 was used. This included the following equipment:

- vibration experiment setup [4] with connections to acquisition board, accelerometer [5], motor [6], motor supply, 5 weights (of 10 g each) and weight holder;
- signal generator Agilent 33220A [7];
- data acquisition board (DAQ) National Instruments NI6115 [8];
- computer (connected to the Internet) with MATLAB software and Data Acquisition Toolbox [9], along with the MQTT [10] and JSON [11] packages.

In the vibration experiment setup, the motor is attached to a stabilization base using foam bumpers. Moreover, the 90 mm diameter wheel fixed to the motor axis with a screw/nut allows to hold weights, causing the axis unbalance. The accelerometer is attached to the motor and allows the user to measure its vibration. The motor supply voltage can be changed in order to change the rotation speed of the motor. Additionally, the connections board CB68-LP (National Instruments) is used to power the accelerometer and to connect the output signals of the three axis to an acquisition board inside the computer (in which the X, Y and Z values are read in channels `ai0`, `ai1` and `ai2`, respectively). It is also important to point out that the acquisition boards NI6220 and NI6229 mentioned in the laboratory guides were not used - instead, NI6115 was utilized, as mentioned above.

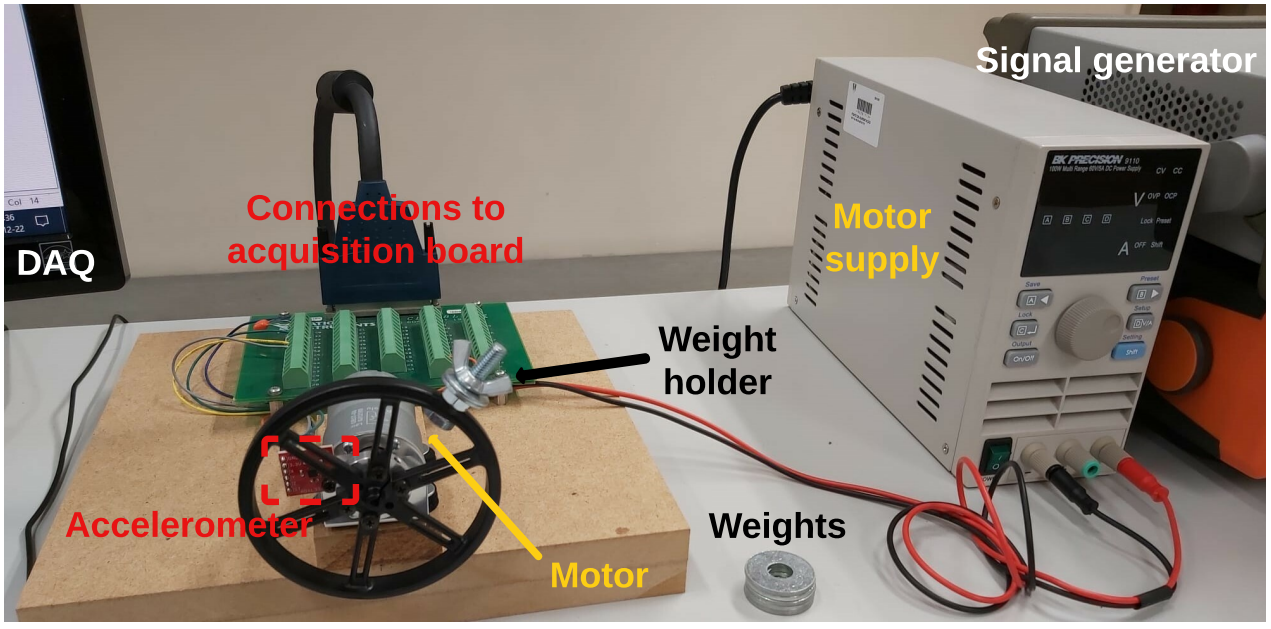

Figure 1 – Experimental setup used in this laboratory assignment.

Initially, the signal generator was directly connected to the DAQ, in order to perform acquisitions using different signals. The former device allows for the generation of sinusoidal, square and triangle waves (among others) with different characteristics - such as frequency, pulse width and amplitude - which makes it suitable for a wide variety of applications requiring a flexible pulse signal.

After this, the DAQ is instead connected to the vibration experiment setup, which contains the accelerometer - available in a lead frame chip scale package. This device allows for 3-axis sensing and can be used in cost-sensitive, low power, motion and tilt-sensing applications. It measures acceleration (such as dynamic acceleration resulting from motion or vibration) with a minimum full-scale range of $\pm 3\ \mathrm{g}$; the user selects the bandwidth using capacitors at the output pins. The output signals are analog voltages that are proportional to the acceleration. In the laboratory setup, the Y-axis was positioned pointing down with a vertical direction, whereas the Z-axis was pointed towards the user (to the front of the setup) and the X-axis was tangent to the wheel (thus, with the same direction as the velocity vector). In the DAQ, there was a minimum sampling rate of $\mathrm{f_s} = 20\ \mathrm{kHz}$ and an input range of $[-2, 2]\ \mathrm{V}$ was used to perform measurements - these informations could be obtained in the MATLAB software.

As for the 12V 37Dx54L 50:1 metal gearmotor, which operates at $12\ \mathrm{V}$ and has a gear ratio of 50:1, it offers approximately the same speed and torque at $12\ \mathrm{V}$ as their $24\ \mathrm{V}$ counterparts do at $24\ \mathrm{V}$, but with approximately double the current draw. With no load, this DC motor should offer a rotation speed of $200\ \mathrm{rpm}$ (i.e., $\approx 3.33\ \mathrm{Hz}$) at maximum voltage. Generally, DC motors can be described by a coil in a magnetic field; when electric current passes through, torque is produced and the motor turns. By considering a constant current, along with equal torque and electrical constants ($\mathrm{k_T}$ and $\mathrm{k_E}$, respectively), the circuit analysis of a typical brushed DC motor leads to the equation

$$\omega = \frac{\mathrm{V}}{\mathrm{k}} - \frac{\mathrm{T}}{\mathrm{k}^2} \cdot \mathrm{R} \Leftrightarrow \mathrm{T} = \frac{\mathrm{V} - \omega \cdot \mathrm{k}}{\mathrm{R}} \cdot \mathrm{k} \tag{1}$$

Where k is a constant and R is a resistance value [12]. This leads to a linear relation between the applied voltage V and the angular velocity of the motor ω - thus, the rotation speed as well. Therefore, when the rotation speed is acquired in the laboratory for a certain supply voltage to the motor, the rotation speeds for other supply voltages can immediately be predicted. Additionally, it can be concluded that the maximum torque T occurs when the angular velocity is null, while there is no load speed for zero torque. In fact, this relation can be seen in Figure 2. Additionally, since the amplitudes A of the signals in the accelerometer are proportional to the acceleration and the centrifugal force is given by $\mathrm{F} = \mathrm{m}\omega^2\mathrm{r}$ (where m is the mass and r the radius), the relations $\mathrm{A} \propto \mathrm{m}$ and $\mathrm{A} \propto \mathrm{V}^2$ are to be expected,

taking into account equation 1 and Newton's Second Law of Motion.

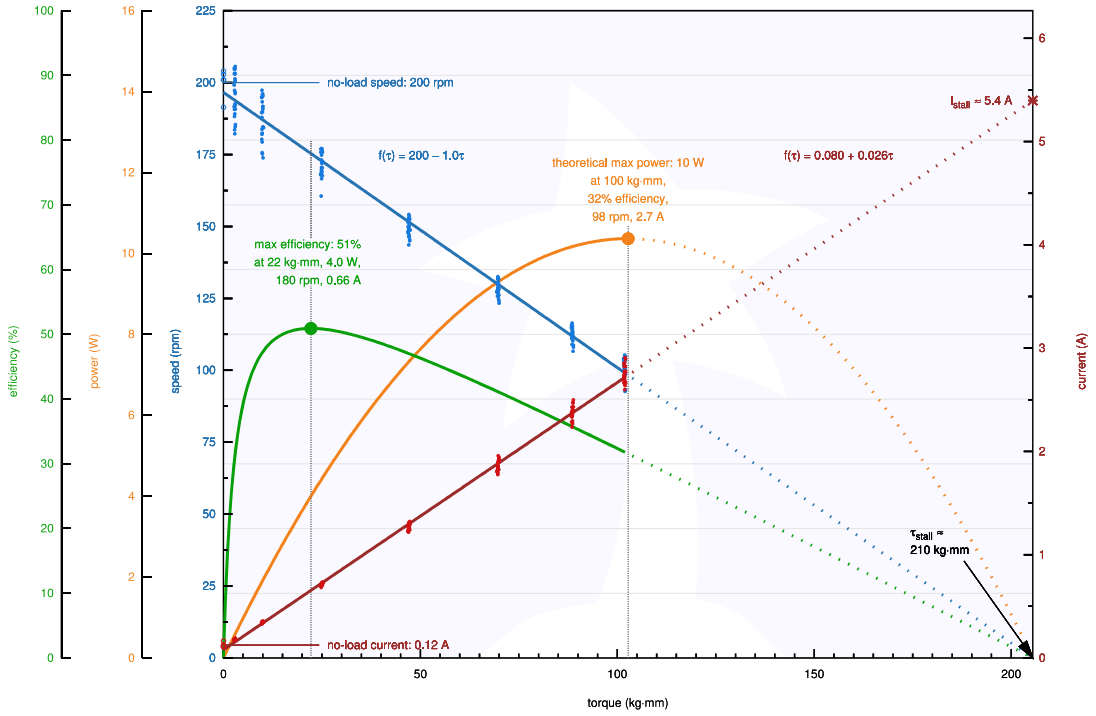

Figure 2 – 12V 37Dx54L 50:1 metal gearmotor performance at 12 V [6].

III. Digital Acquisition and Processing

III.1 Single-sided spectrums

Initially, the MATLAB script `'step1_step2.m'` (shown in Appendices) was developed to compute the Discrete Fourier Transform of an acquired signal and display its single-sided spectrum, using the Data Acquisition Toolbox. Using this code, the signal generator was connected to the DAQ and signals with different characteristics were used to obtain the results shown in Figures 3 to 6, in which the respective (rounded) apparent frequencies f are also shown.

In the results shown in Figure 3, no aliasing occurs, since $f_s = 100$ kHz $> 2f = 2$ kHz, thus the signal frequency and apparent frequency are the same. No significant change in the apparent frequency occurs with the change in amplitude, since this is a phenomenon mostly dependant on frequency. However, more spread occurred around the frequency value. A range of $[-10, 10]$ V was selected in the DAQ to acquire the signal of 4 V in amplitude. Since $f_s/f = 100$, 100 points per period are recovered, the sine waves are flawlessly replicated and the FFT has a single peak in 1000 Hz.

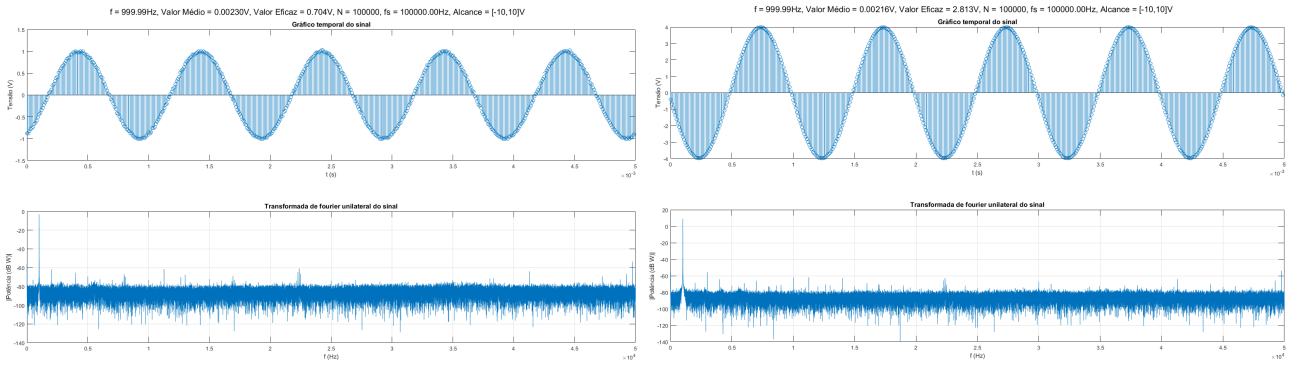

Figure 3 – Acquired signal and single-sided spectrum for an input sinusoidal waveform of frequency $\mathrm{f} = 1$ kHz and amplitude 1 V [left] or 4 V [right], using 100 000 acquisition points and the sampling rate $\mathrm{f_s} = 100$ kHz.

When the number of acquisition points changes, once again, aliasing does not occur, since $\mathrm{f_s} > 2\mathrm{f}$, thus the input frequency and apparent frequency are the same. However, a significantly higher spread ("espalhamento") occurs when $999\ 950$ points are used. Since the period of the input signal is 1 ms and $1/\mathrm{f_s} = 10\ \mu$s, only full periods are recovered with $100\ 000$ points. However, with $999\ 950$ points, the relation $\frac{999\ 950 \times 10\ \mu\mathrm{s}}{1\ \mathrm{ms}} = 9999.5 \notin \mathbb{Z}$ is true, thus the last period is not completely sampled. This higher number of points leads, on the other hand, to a higher frequency resolution (i.e., lower Δf). It is worth noting that the IpDFT (Interpolated DFT) used in `'step1_step2.m'` should help the apparent frequency become more similar to the signal frequency - with a different algorithm, perhaps a more significant difference could be seen. Moreover, an RMS value ("Valor Eficaz") of 0.704 V was obtained for both cases, since only complete periods were considered for its calculation.

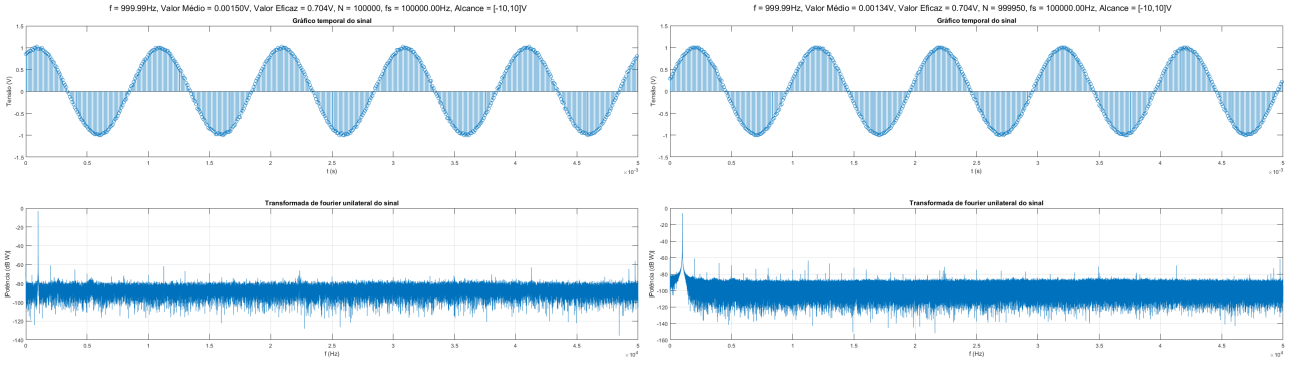

Figure 4 – Acquired signal and single-sided spectrum for an input sinusoidal waveform of frequency $\mathrm{f} = 1$ kHz and amplitude 1 V, using 100 000 [left] or 999 950 [right] acquisition points and the sampling rate $\mathrm{f_s} = 100$ kHz.

In the first three cases shown in Figure 5, the signal is sampled at twice or above its frequency, thus it can be recuperated (according to the Digital Sampling Theorem). With 60 kHz and 100 kHz this does not occur, leading to spectral aliasing ("espelhamento"). The input signal spectrum is reduced to frequencies from 0 to $\mathrm{f_s}/2$, thus the input spectrum from $\mathrm{f_s}/2$ to $\mathrm{f_s}$ appears mirrored from 0 to $\mathrm{f_s}/2$. With 60 kHz, it occurs that $\mathrm{f_s}/2 - (60\ \mathrm{kHz} - \mathrm{f_s}/2) = 40$ kHz, thus this value was obtained for the

apparent frequency. A similar thinking can be applied to 100 kHz, with which $f_s/f = 1$, therefore one point per period - in the same voltage value - is recovered and the apparent frequency is 0 Hz (same of a DC signal). In the second case (with $f = 40$kHz), it is also worth noting that $f_s/f \in \mathbb{Q},\ \notin \mathbb{N}$, therefore the samples of each period do not correspond to the same phase of the input signal; however, after a certain number of periods, they start repeating.

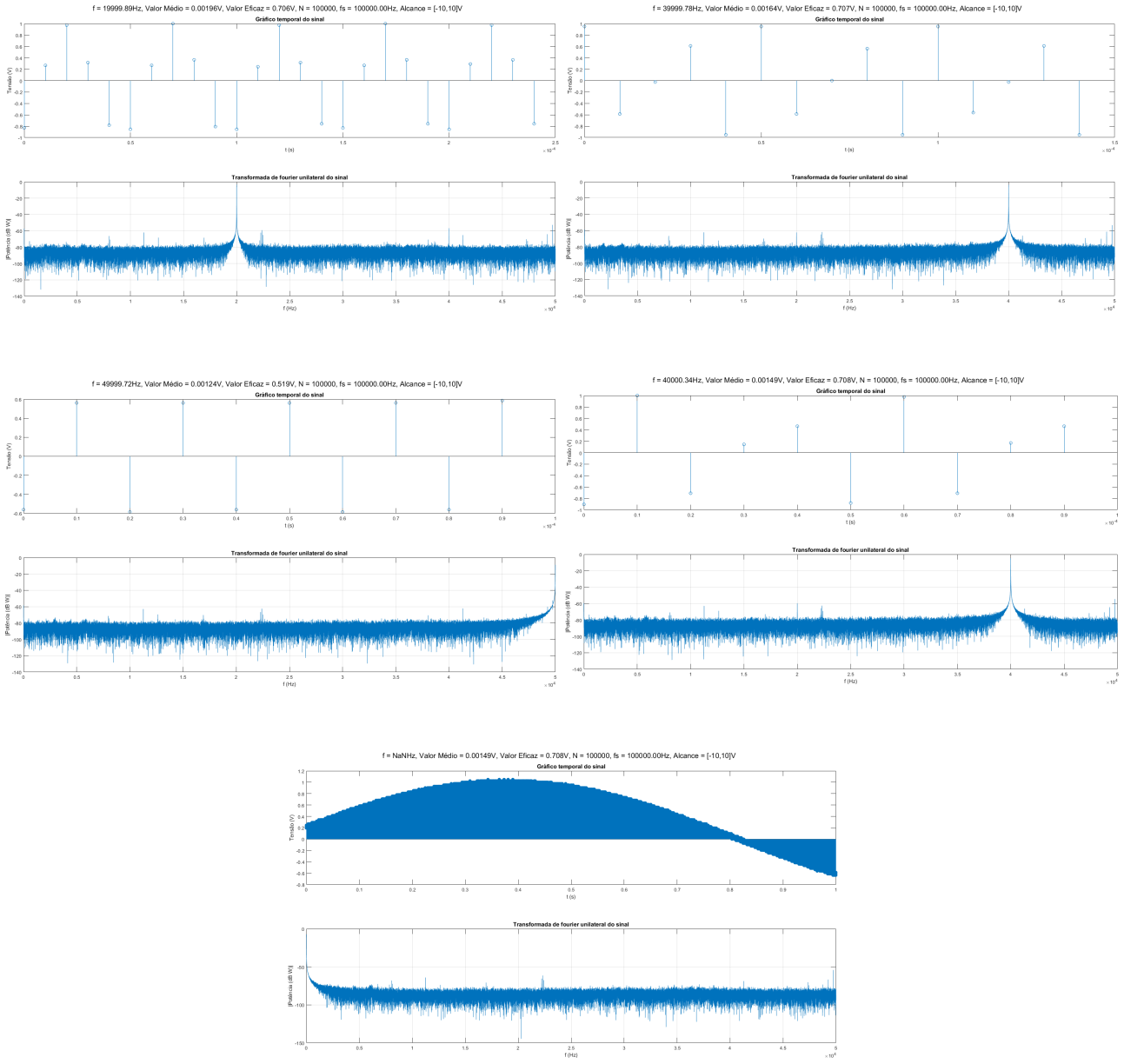

Figure 5 – Acquired signal and single-sided spectrum for an input sinusoidal waveform of frequency $f = 20$ kHz [top left], $f = 40$ kHz [top right], $f = 50$ kHz [middle left], $f = 60$ kHz [middle right] or $f = 100$ kHz [bottom] and amplitude 1 V, using 100 000 acquisition points and the sampling rate $f_s = 100$ kHz.

The Fourier series of a triangle wave is given by

$$f(x) = \frac{8}{\pi^2} \sum_{n=1,3,5,\ldots}^{\infty} \frac{(-1)^{\frac{n-1}{2}}}{n^2} \sin \frac{n\pi x}{L} \tag{2}$$

So the harmonics have amplitudes which decrease with $1/n^2$, becoming less significant for higher frequencies, thus the apparent frequency is ≈ 1 kHz. For the square wave,

$$f(x) = \frac{4}{\pi} \sum_{n=1,3,5,\ldots}^{\infty} \frac{1}{n} \sin \frac{n\pi x}{L} \tag{3}$$

Therefore the same thinking can be applied. However, the amplitudes now decrease with $1/n$, so an apparent frequency marginally different from 1 kHz is obtained. Since the triangle and square waves are infinite series, aliasing will always occur. This is very clear with an 15 kHz signal frequency, where peaks appear at 5, 15, 25, 35 and 45 kHz. What seems to be spread in the spectrum is also clear around these frequencies. However, this should be mostly due to the aliasing phenomenon as well, which ends up occurring at successively slightly deviated frequency values. Additionally, it can be seen that other significant peaks in the spectrum occur at 10, 20, 30, 40 and 50 kHz, which should be due to even number harmonics in the square wave (for instance, due to a duty cycle different from 50%).

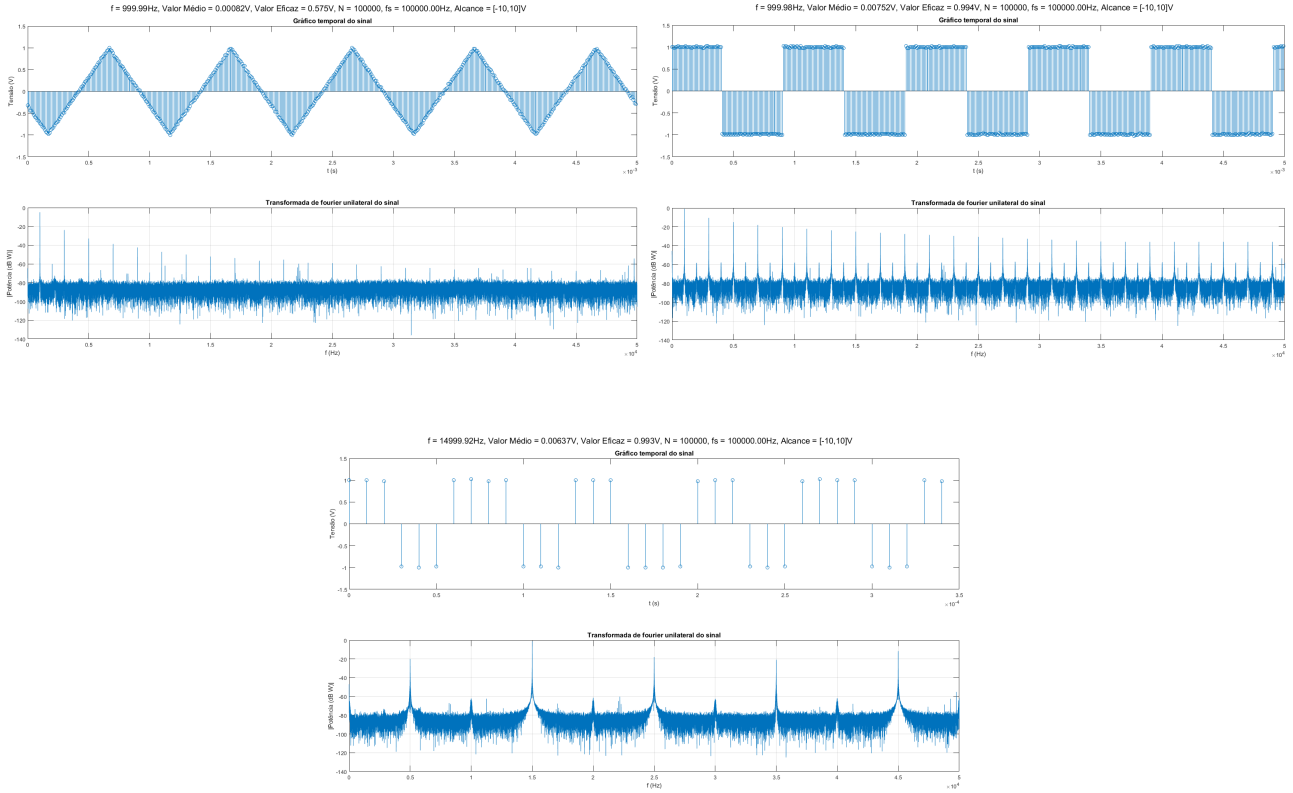

Figure 6 – Acquired signal and single-sided spectrum for an input triangle [top left] or square [top right, bottom] waveform of frequency $f = 1$ kHz [top] or $f = 15$ kHz [bottom] and amplitude 1 V, using 100 000 acquisition points and the sampling rate $f_s = 100$ kHz.

III.2 Mean and root mean square values

After connecting the vibration experiment setup to the DAQ, acquisitions were initially performed without the motor being powered, from which the mean value for each axis signal was obtained. Once the motor had been connected to the power supply set to $12\ \mathrm{V}$, another acquisition was performed and the root mean square value for the X-axis signal was obtained. These results are shown in Table 1.

The values of $\mathrm{Mean\ (X)}$ and $\mathrm{Mean\ (Z)}$ are more similar to each other, while the value $\mathrm{Mean\ (Y)}$ is significantly lower. This is because of the axis definitions: $\overrightarrow{y}$ points downwards, $\overrightarrow{x}$ is tangent to the surface and $\overrightarrow{z}$ points to the front of the setup. Since no rotation is occurring, only the weight is significant, thus altering $\mathrm{Mean\ (Y)}$. In all three cases, the same RMS values were obtained (i.e., $\mathrm{RMS\ (X) = Mean\ (X)}$, $\mathrm{RMS\ (Y) = Mean\ (Y)}$ and $\mathrm{RMS\ (Z) = Mean\ (Z)}$), since these are DC signals. In fact, very low frequencies of $7.98\ \mathrm{Hz}$, $15.36\ \mathrm{Hz}$ and $0.36\ \mathrm{Hz}$ were respectively measured using `'step1_step2.m'`. To obtain these values, a sampling frequency of $\mathrm{f_s} = 20\ \mathrm{kHz}$ and a number of points of $\mathrm{N} = 200\ 000$ were used. Next, it is important to determine the RMS values of the AC components, which are given by

$$\mathrm{RMS_{AC}} = \sqrt{\mathrm{RMS^2} - \mathrm{RMS_{DC}^2}} \tag{4}$$

Where $\mathrm{RMS_{AC}} = \mathrm{RMS[X - Mean(X)]}$ and $\mathrm{RMS_{DC}} = \mathrm{Mean(X)}$ for the parameter X. By computing the RMS values for each component, while the motor was rotating, very low values were obtained for $\mathrm{RMS(Y - \overline{Y})}$ and $\mathrm{RMS(Z - \overline{Z})}$, whilst more significant measurements were obtained for the X-axis signal. This makes sense since $\overrightarrow{x}$ has the same direction as the velocity $\overrightarrow{v}$ (tangent to the surface) and is therefore affected by the centrifugal force $\mathrm{F} = \mathrm{m}\omega^2\mathrm{r}$. Because of this, throughout the following measurements, only values for the X-axis are obtained (i.e., using channel 'ai0' in the setup).

Table 1: Vibration mean value for each axis signal (without the motor being powered), root mean square value for the X axis signal and root mean square value for the AC component of the X axis signal (using a supply voltage of 12 V).

Mean (X) [V]	**Mean (Y) [V]**	**Mean (Z) [V]**	**RMS (X) [V]**	**RMS[X-MEAN(X)] [V]**
1.57433	1.29411	1.65387	1.667094	0.54835

In order to upgrade this experimental setup, an OP-AMP could be used to amplify the signal around $\mathrm{Mean\ (X)} \approx 1.57\ \mathrm{V}$ (to fix the low range $[-2, 2]\ \mathrm{V}$ used in the DAQ). A capacitance of $1\ \mu\mathrm{F}$ is already used to filter high frequencies, but a higher capacitance could be used to filter DC and very low frequencies as well. This capacitance value would not need to be that high, since the output resistance of the OP-AMP should already be very considerable.

III.3 Motor rotation speed and unbalancing level

Using a different number of unbalancing weights and different supply voltage values, multiple measurements were performed and the data from each acquisition was saved in a separate `.mat` file. A sampling rate of $\mathrm{f_s} = 20\ \mathrm{kHz}$ was used since it was the minimum value allowed by the DAQ. In order to obtain a resolution of $0.05\ \mathrm{Hz}$, the number of points was then selected. An even lower $\Delta\mathrm{f}$ could have been used (which could have led to less spread in the spectrum later on), but this would lead to higher acquisition times. With these acquisition parameters, each acquisition takes 20 seconds. Since the OP-AMP configuration mentioned before was not actually used, the low input range of $[-2, 2]\ \mathrm{V}$ still applies. For this reason, three acquisitions are performed each time, leading to a total "actual" acquisition time of 1 minute; after this, an average is performed. This way, it is possible to considerably decrease the noise floor. Alternatively, however, it could have been decided to perform only one acquisition at a time, but using a higher number of points N. In fact, using a larger N provided generally better mass predictions in section IV. On the other hand, a big advantage of using a lower N value and performing the average every 3 acquisitions is that new results are displayed with `'step10.m'` (and in the ThingSpeak platform) more frequently. It is also worth noting that the minimum $\mathrm{f_s} = 20\ \mathrm{kHz}$ is not a problem since the relevant frequencies in this assignment will only go from just above $0\ \mathrm{Hz}$ to $\approx 3.56\ \mathrm{Hz}$ (for $12\ \mathrm{V}$).

Table 2: Acquisition parameters used in the algorithms to estimate the motor rotation speed and unbalancing level.

Input range [V]	**Sampling rate, $\mathrm{f_s}$ [kHz]**	**Number of points**
[-2, 2]	20	400 000

Having obtained measurements for all weights and all supply voltages, the spectrograms in Figures 7 and 8 are given for $12\ \mathrm{V}$ and 5 weights, respectively, since the results are clearer in these cases. For this purpose, the script `'step6.m'` (included in Appendices) was created. This script loads the successive files obtained previously and creates the spectrograms using the `imagesc` function in MATLAB. The frequency values are estimated with an IpDFT (Interpolated Discrete Fourier Transform), whereas the amplitudes (in watts) correspond to the highest peak amplitudes in the single-sided spectrums.

In the operation of a DC motor, the rotating frequency varies linearly with the voltage. Since, for $12\ \mathrm{V}$, a frequency of $3.56\ \mathrm{Hz}$ was obtained, the frequency for a voltage x is given by $(\mathrm{x}/12) \cdot 3.56\ \mathrm{Hz}$. In Figure 8, a linear relation is apparent, although not for $4\ \mathrm{V}$ and $6\ \mathrm{V}$, since a step of $1\ \mathrm{V}$ was used for higher supply voltages (i.e., voltages of $4,\ 6,\ 8,\ 9,\ 10,\ 11$ and $12\ \mathrm{V}$ were selected). Moreover, it is also clear that the amplitude of the peak varies according to $\mathrm{A} \propto \mathrm{V}^2$ - as mentioned in section II. In Figure 7, it is clear that $\mathrm{A} \propto \mathrm{m}$ and the peaks occur in the same frequency, since the supply voltage is

constant. The other less relevant peaks seen in these images (with the same spacing from each other) are due to the Fourier series of the signals.

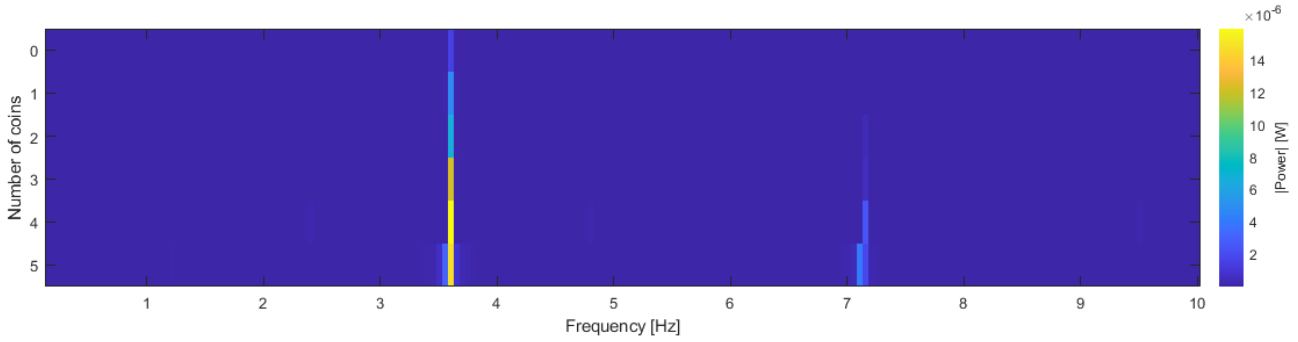

Figure 7 – Spectrogram for the unbalancing weight variations (0 to 5), using a supply voltage of 12 V.

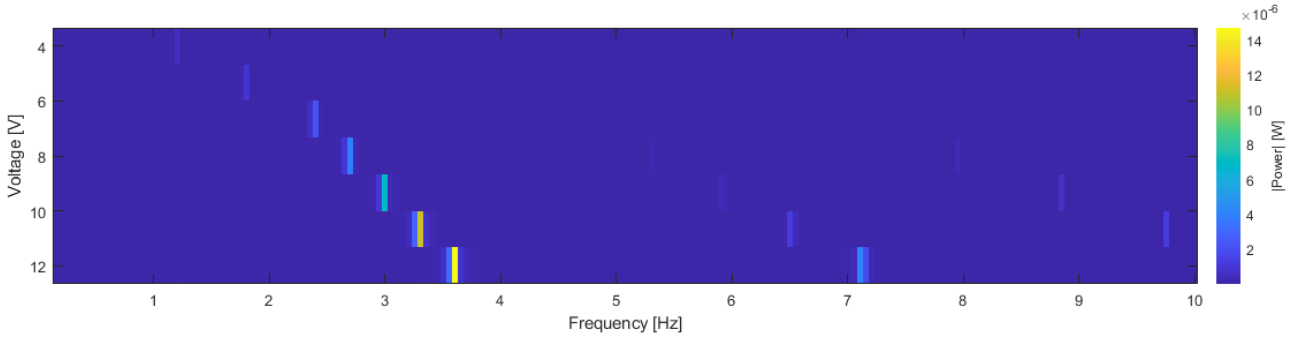

Figure 8 – Spectrogram for the supply voltage variations (4, 6, 8, 9, 10, 11 and 12 V), using 5 unbalancing weights.

The rotation speed is given by $\mathrm{CP} \times 60\ \mathrm{min}$ in rpm, where $\mathrm{CP}\ [\mathrm{Hz}]$ is the frequency estimated by IpDFT. For this purpose, a rectangular window around the maximum amplitude value of the DFT is used, having the maximum values neighboring this peak been calculated for this purpose (as well as the respective frequencies). To estimate the unbalancing weight (i.e., the weight put on the vibration setup), the regression parameters shown below (equation 6) should be used. This equation will then allow users to know an undetermined mass in the setup, having first computed the values of the rotation frequency (CP) and peak amplitude ($|\mathrm{CP}|$). As mentioned before, decided to use the peak amplitude in watts (W), since it provided better results for the mass estimations. It would also be possible to check the amplitude in the estimated frequency CP, although spread in the spectrum could compromise the results in that case.

To obtain the 3D plot shown in Figure 9, the function `polyfitn` was used, along with `polyvaln` (as implemented in `'step8_step9.m'`, included in Appendices). An incredibly satisfactory fit was obtained with a polynomial of order 3. With an order of 2, a similar behaviour was seen for higher frequencies, but the plot varied more significantly for lower CP. With an order of 4, good results were also obtained, but the values of the curve start decreasing for higher CP values. A polynomial of order

5 led to a sharp increase in the curve values for low CP and a sharp decrease for high CP. Thus, an order of 3 seemed to provide the most satisfactory fit, in which the previously mentioned relations $|\mathrm{CP}| \propto \mathrm{CP}^2$ and $|\mathrm{CP}| \propto \mathrm{m}$ can be seen. The function `polyn2sympoly` can then be used to obtain the regression parameters shown in equations 5 and 6.

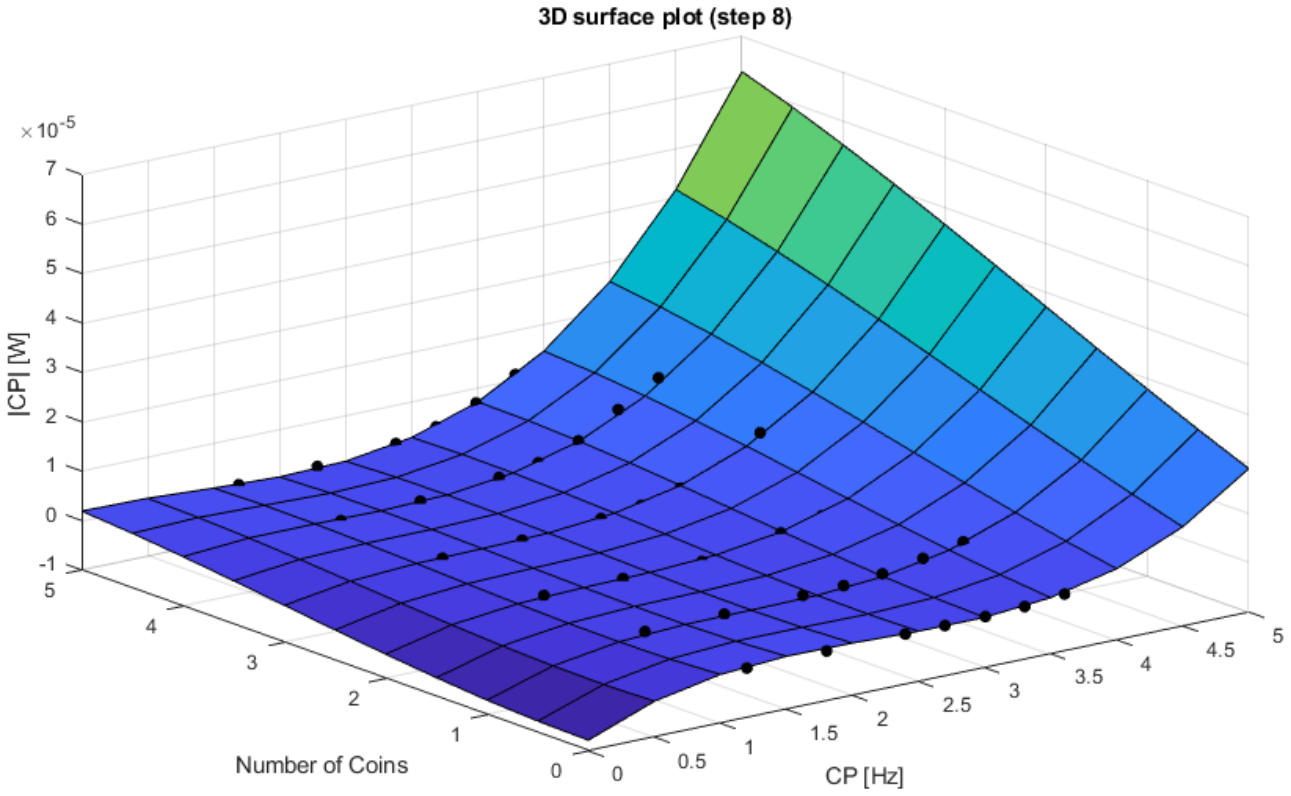

Figure 9 – 3-axis plot for the number of unbalancing weights, rotating frequency (CP) and highest peak amplitude in the single-sided spectrum (|CP|). The functions `polyfitn` and `polyvaln` were used to obtain the regression surface plot for the unbalancing weight estimation, which corresponds to a polynomial of order 3 with variables CP and |CP|.

Applying the aforementioned script, regressions were performed to allow estimating the rotation speed - which can be used in case the mass is known - and the unbalancing weight - in case the frequency is known. The former is given by

$$\begin{aligned}\mathrm{CP[Hz]} = {} & (2.44\times10^{15})\mathrm{x}_1^3 + (2.34\times10^{8})\mathrm{x}_1^2\mathrm{x}_2 - (7.01\times10^{10})\mathrm{x}_1^2 + (1.31\times10^{4})\mathrm{x}_1\mathrm{x}_2^2 - \\ & -(9.59\times10^{4})\mathrm{x}_1\mathrm{x}_2 + (8.15\times10^{5})\mathrm{x}_1 - (8.30\times10^{-3})\mathrm{x}_2^3 + (8.41\times10^{-2})\mathrm{x}_2^2 - \\ & -(4.54\times10^{-1})\mathrm{x}_2 + 2.21\end{aligned} \tag{5}$$

Where $\mathrm{CP} = \mathrm{f}(|\mathrm{CP}|, \mathrm{m})$, with $\mathrm{x}_1[\mathrm{W}] \equiv |\mathrm{CP}|$ and $\mathrm{x}_2[\text{number of coins}] \equiv \mathrm{m}$. The consequent rotation speed in rpm is given by $\mathrm{CP[rpm]} = \mathrm{CP[Hz]} \times 60\ \mathrm{min}$. The more relevant regression $\mathrm{m} = \mathrm{f}(\mathrm{CP}, |\mathrm{CP}|)$ ($\mathrm{x}_1 \equiv \mathrm{CP}, \mathrm{x}_2 \equiv |\mathrm{CP}|$) - used later on in the ThingSpeak platform - is given, in the same units as before, by the equation

$$\begin{aligned}\mathrm{m[number\ \ of\ \ coins]} = {} & (-3.04\times10^{-1})\mathrm{x}_1^3 + (1.36\times10^{6})\mathrm{x}_1^2\mathrm{x}_2 + 1.79\mathrm{x}_1^2 - \\ & -(4.89\times10^{10})\mathrm{x}_1\mathrm{x}_2^2 - (9.03\times10^{6})\mathrm{x}_1\mathrm{x}_2 - 3.61\mathrm{x}_1 + (9.17\times10^{14})\mathrm{x}_2^3 + \\ & +(1.45\times10^{11})\mathrm{x}_2^2 + (1.55\times10^{7})\mathrm{x}_2 + 2.98\end{aligned} \tag{6}$$

Having the coefficients been rounded in both regression functions. Since each weight corresponds to 10 g, the mass estimates in grams are obtained by multiplying equation 6 by this value.

Finally, the MATLAB script `'step10.m'` was coded to continuously acquire the X-axis vibration, compute the necessary features and display the single-sided spectrum, along with the rotation speed and unbalancing weight estimate values. Once again, it is worth mentioning that an order of 3 in the polynomials was used - since it also produced better results with this code. Higher orders lead to over-fitting, while an order of 2 did not adjust as well to the data points. As mentioned before, the amplitude $|\mathrm{CP}|$ (given by $(\mathrm{Voltage\ value})/\sqrt{2})^2$) is expressed in watts, since power values provided better results with `'step10.m'`, having these amplitudes been determined at the maximum peak amplitude. However, while running this script, some spread in the spectrum was evident, thus the possibility of considering the one or two nearest neighbouring frequency(ies) on each side to calculate CP was considered, but did not upgrade the quality of the results.

IV. Cloud Sensor Platforms

By recalling the code implemented `'step10.m'` mentioned in section III, three new MATLAB scripts (all included in Appendices) were developed to continuously acquire, compute and report the feature values (frequency and peak amplitude) to the ThingSpeak platform. In `'step1.m'`, the features are reported using `thingSpeakWrite`. In `'step2.m'`, the MQTT package is used; due to the `num2str` restriction and the scientific notation in the (low) values of the amplitude, it is crucial to change from scientific notation to `long` format (for example), as implemented in the code. Finally, `webwrite` in `'step3.m'` uses JSON and MGSPACK formats.

Initially, the `'step3.m'` script was used to compare the size and readability of the JSON and MGSPACK payloads. A size of 87 bytes was determined for the JSON payload size, along with a smaller 62 bytes for the MSGPACK payload. JSON (JavaScript Object Notation) is a lightweight data-interchange format, completely language-independent. In JSON, an object begins with "{" and ends with "}"; each name is followed by ":" and the pairs are separated by ",". This could be seen when the MATLAB data structure with the API key, field 1 and field 2 values were converted to a JSON MessagePack formatted string. The latter is a compacted binary JSON-like format, in which 'struct',

'string' and 'scalar' (from MATLAB) are converted to 'map', 'string' and 'number', respectively. This efficient binary serialization format allows for a smaller payload size. However, it is not directly readable and cannot be sent to ThingSpeak, as opposed to the JSON payload. In the MSGPACK format, it was seen that the strings from the MATLAB structure were codified the same way, but the values of field 1 and field 2 were not, since fewer bytes are used to represent them.

In the ThingSpeak platform, two channels were created: a private channel (Channel ID: 2004335) and a public channel (Channel ID: 2000986). The former receives the values of the frequency and peak amplitude (in Hz and W, respectively) and plots them (as indicated in Figure 10), whereas the rotation speed and the unbalancing weight estimation (in rpm and grams, respectively) are plotted in the latter - as shown in Figure 12. In the public channel, lamp indicators (shown in Figure 13) are turned on when the rotation speed exceeds 20 rpm and the unbalancing weight exceeds 25 g, respectively. The frequency and peak amplitude are processed in MATLAB Analysis - in which $\text{Speed} = \text{Frequency} \times 60$ and Weight (given by the regression equation 6) are calculated, as shown in Figure 11.

As opposed to the measurements present in section III, $\text{N} = 1\ 000\ 000$ acquisitions were now performed (instead of $\text{N} = 400\ 000$), since the platform had a computing timeout of 20 seconds. With this new N value, a frequency resolution of $\Delta\text{f} = 0.02$ Hz is used, along with an acquisition time of 1 minute. In fact, this new configuration led to more satisfactory results in the unbalancing weight predictions, when compared to the values obtained with `'step10.m'`. Thus, the procedures from section III could have been repeated with this new frequency resolution to obtain better results with the regression equation.

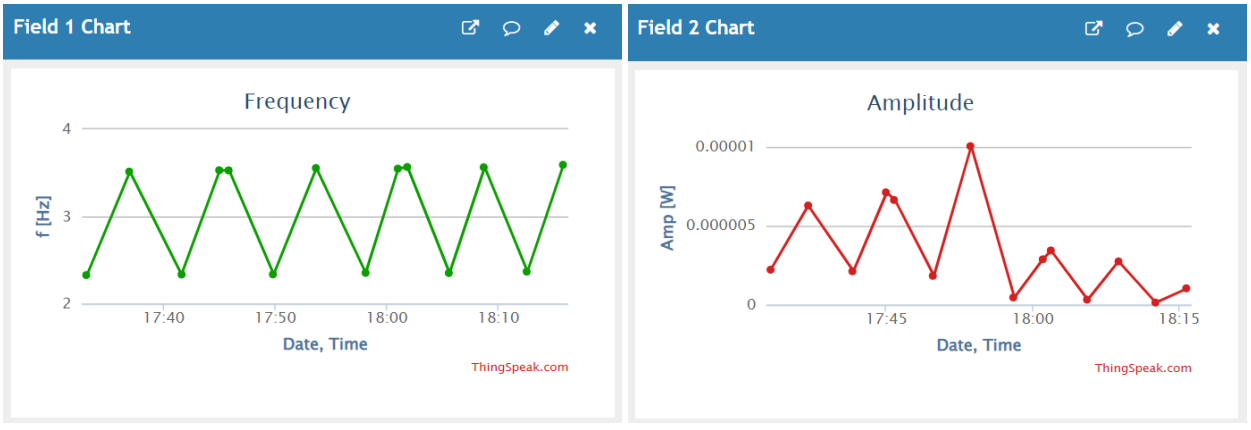

Figure 10 – Chart indicators for the rotation frequency and peak amplitude in the spectrum, implemented in the private channel of the ThingSpeak platform.

Name

Speed and Weight calculations

MATLAB Code

```
% Channel ID to read data from
readChannelID = 2004335;
% Frequency Field ID
FrequencyFieldID = 1;
% Amplitude Field ID
AmplitudeFieldID = 2;

% Channel Read API Key
% If your channel is private, then enter the read API Key between the '' below:
readAPIKey = 'K9KX90V94JQHLNTG';

Freq = thingSpeakRead(readChannelID,'Fields',FrequencyFieldID,'ReadKey',readAPIKey);
Amp = thingSpeakRead(readChannelID,'Fields',AmplitudeFieldID,'ReadKey',readAPIKey);
disp(Freq)
disp(Amp)

Weight = (-0.34079*Freq^3 + 1410971.3788*Freq^2*Amp + 2.0134*Freq^2 - 62354228496.1557*Freq*Am
Speed = Freq*60;
disp(Weight)
% Replace the [] with channel ID to write data to:
writeChannelID = [2000986];
% Enter the Write API Key between the '' below:
writeAPIKey = '984YKO53M0U0HRU6';

thingSpeakWrite(writeChannelID,{Weight, Speed},'WriteKey',writeAPIKey)
```

Save and Run Save

Figure 11 – ThingSpeak MATLAB Analysis code (executed in the platform itself) to post-process the reported frequency and amplitude features and estimate the motor rotation speed and unbalancing weight.

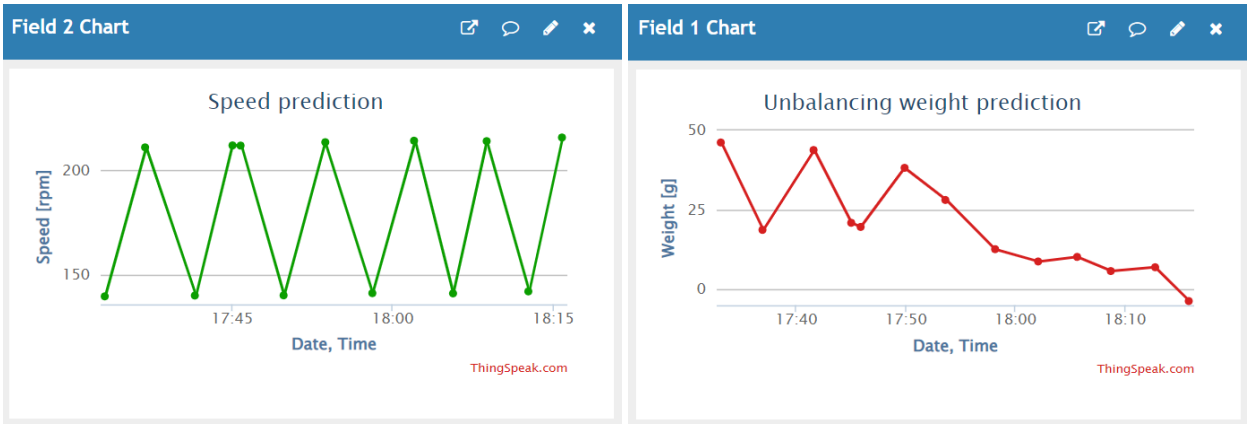

Figure 12 – Chart indicators for the rotation speed and unbalancing weight estimates, implemented in the public channel of the ThingSpeak platform.

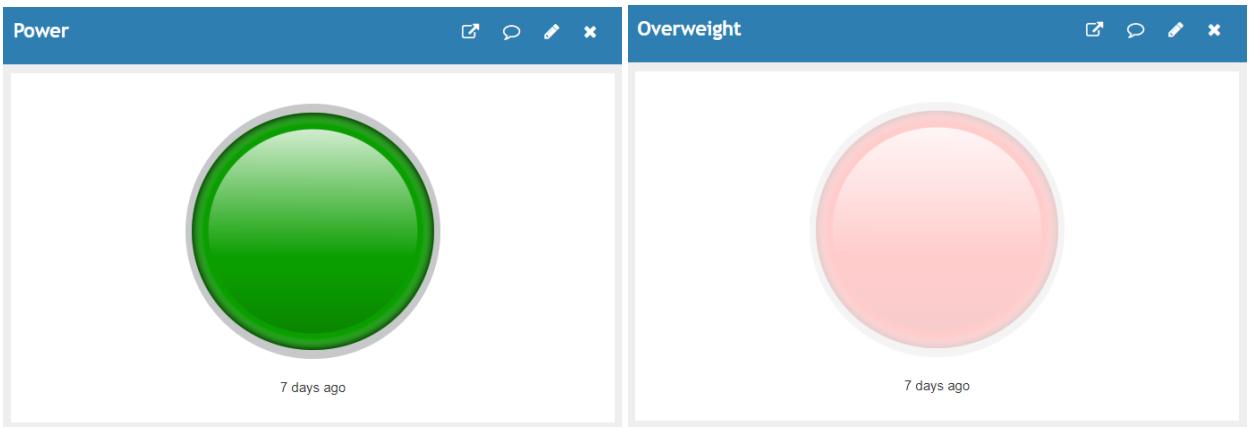

Figure 13 – Lamp indicators that specify if the motor rotation speed rises above 20 rpm [left] or the unbalancing weight is higher than 25 g [right], implemented in the public channel of the ThingSpeak platform.

The results shown in the ThingSpeak platform (Figures 10 to 13) were obtained with `'step1.m'`, for supply voltages of 8 V and 12 V and 0 to 5 weights. Chronologically, data was obtained for: 5 masses and 8 V, 5 masses and 12 V, 4 masses and 8 V, 4 masses and 12 V, ..., until 0 masses and 12 V. In total, 12 points are shown in Figure 12 (+1, since two points were acquired for 4 masses and 12 V, by accident). These results are included in Table 3. The rotation speed varied between close to 214 rpm (for 12 V) and 142 rpm (for 8 V), as expected. With 12 V, worse weight predictions were generally obtained, since the 3D curve (in Figure 9) has very high derivative values when the frequency is too high - this is not the case for 8 V. The only case in which 8 V provides a worse weight prediction is for 0 g, since with 0 coins there is a more significant decline in the 3D curve for lower frequencies. With 8 V, the masses 50 g, 40 g and 10 g were correctly predicted. In general, however, it can be seen that this work allows the user to obtain very satisfactory mass predictions when the supply voltage is not raised too high. However, mass predictions are also limited by the vibration experiment setup itself, which was confirmed in the laboratory to not be completely reliable. In fact, performing measurements with the same supply voltage and unbalancing weight led to successively different values over time - which compromises the use of this equipment for this type of analysis.

Table 3: Results displayed in the ThingSpeak public channel chart indicators, for a different number of unbalancing weights (0 to 5) and supply voltages of 8 V and 12 V. The rounded mass estimates are also shown, along with the respective expected values, considering a mass of 10 g for each unbalancing weight.

Number of weights	Supply voltage [V]	Speed [rpm]	Mass [g]		
			Experimental	Rounded	Expected
5	8	139.7	46.2	50	50
	12	210.8	18.7	20	
4	8	140.0	43.8	40	40
	12	211.8	20.9	20	
3	8	140.1	38.2	40	30
	12	213.3	28.1	30	
2	8	141.2	12.6	10	20
	12	214.0	8.8	10	
1	8	141.1	10.2	10	10
	12	213.8	5.7	10	
0	8	142.0	6.9	10	0
	12	215.6	-3.7	0	

In the final step of this procedure, the ThingView Free mobile app was installed on an Android device; the public and private channels were then associated to the app, as shown in Figure 14.

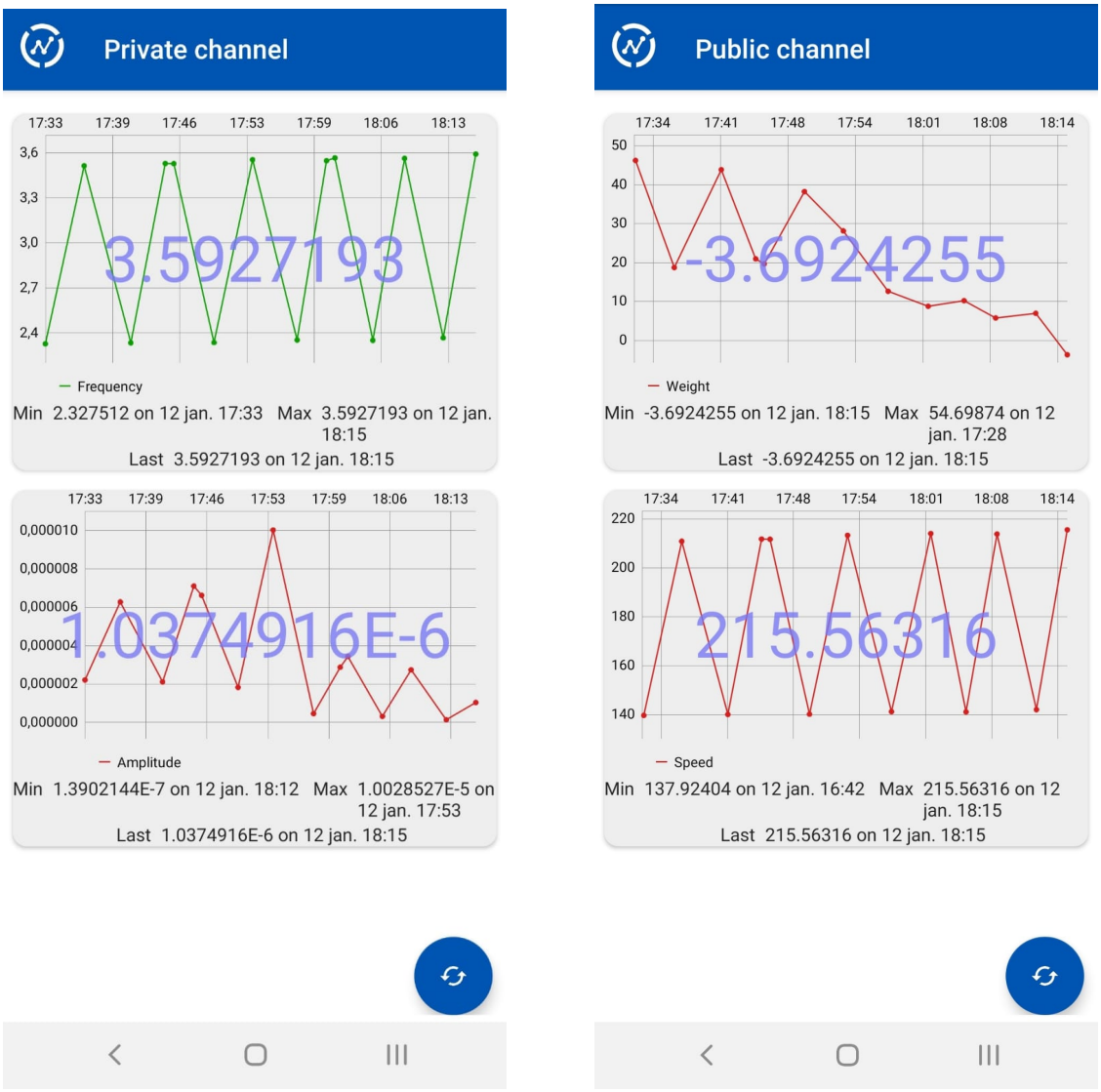

Figure 14 – Private channel and public channel chart indicators included in the ThingView Free mobile app.

V. Conclusion

In this laboratory assignment, all of the proposed objectives have been achieved. Initially, single-sided spectrums were computed for different input signals and the respective results were analysed taking into account the aliasing phenomenon. Using the vibration experiment setup, the signal in the X-axis was acquired for different supply voltage values and unbalancing weights, from which the resulting spectrograms proved the expected relations between frequency, amplitude and voltage. Then, a regression was performed with a 3D surface to obtain an equation that allows the user to obtain estimates of the unbalancing weight. Using the ThingSpeak platform, the rotation speed was correctly predicted for different configurations, whereas the mass predictions corresponded to the expected values in 50% of the data points. However, some measurements had been obtained with 12 V of supply voltage to purposely test the limits of the regression equation.

On a final note, having a complete picture of the project obtained, some improvements to the procedure could be executed. Due to ThingSpeak platform having a computing timeout period of 20 seconds, the mass estimation function was used with a higher N (number of acquisition samples), which improved the results, since the acquisition time rises above 20 seconds and the DFT has a better resolution - thus, a similar procedure could be done to perform all measurements once again. However, even though the higher acquisition time eliminates most of the spread, correct mass estimates were still not obtained for all tested supply voltage and number of unbalancing weights combinations. Furthermore, to help neglect the spread, the calculation of $|\mathrm{CP}|$ could be tweaked. Instead of being the highest peak amplitude in the interval $[0.1, 4.0]$ Hz, it could be considered as this peak amplitude summed with the amplitudes that surround it and are within less than -20 dBW of it.

References

[1] Lab 2 Guide, Big-Data Measuring Systems website, Luís Rosado, IST, 2021

[2] Lab 3 Guide, Big-Data Measuring Systems website, Luís Rosado, IST, 2021

[3] S. Lacey, "The Role of Vibration Monitoring in Predictive Maintenance", white paper, Schaeffer Limited (UK)

[4] Vibration Analysis Experiment Setup, Big-Data Measuring Systems webpage, MECD, 2020

[5] Accelerometer ADXL337 datasheet, Big-Data Measuring Systems webpage, MECD

[6] 37D Metal Gearmotors datasheet, Big-Data Measuring Systems webpage, MECD

[7] Agilent 33220A 20MHz Funtion/Arbitrary Waveform Generator datasheet, Agilent Technologies, USA, 2004

[8] NI 6115/6120 Specifications, National Instruments, USA, 2018

[9] MATLAB Data Acquisition Toolbox User's Guide, Big-Data Measuring Systems webpage, MECD

[10] MATLAB MQTT package, Big-Data Measuring Systems webpage, MECD

[11] MATLAB JSON package, Big-Data Measuring Systems webpage, MECD

[12] "The relationship between voltage and dc motor output speed", Motion Control Tips, Danielle Collins, 2015

Appendices

step1_step2.m

```
fs = 20e3; % Acquisition rate
N = 400e3; % Number of acquisitions
TOTAL_ACQ = 3; % Process repeated 3 times, average computed

% -----visualização do DAQ-----
dispositivo = daqlist;
placa_ID = dispositivo.DeviceID;
placa_vendedor = dispositivo.VendorID;
deviceInfo = dispositivo{1, "DeviceInfo"};

% -----Programação do DAQ-----
dq = daq(placa_vendedor);
dq.Rate = fs;
ch = addinput(dq, placa_ID, 'ai0', 'Voltage');
ch.Range=[-10,10];
fs = dq.Rate;

% ----Obter sinal com DAQ----
start(dq, "Continuous")
for i=1:TOTAL_ACQ
    [signal(:, i), t(:, i)] = read(dq, N, "OutputFormat", "Matrix");
    fprintf("Foram obtidas %d amostras do sinal [%d/%d].\n", N, i, TOTAL_ACQ);
end
stop(dq)

% Resoluções e fs reais
dt = t(2)-t(1);
fs = 1/dt;
df = fs/N;
disp(['Frequencia de amostragem = ', strtrim(evalc('disp(double(fs))')), ' Hz'])
disp(['Resolução Temporal = ', strtrim(evalc('disp(double(dt))')), ' s'])
disp(['Resolução Espectral = ', strtrim(evalc('disp(double(df))')), ' Hz'])
fprintf("Aquisição irá demorar %d s\n", dt*N*TOTAL_ACQ);

% Estimar a frequência
% Transformada de Fourier
for i = 1:TOTAL_ACQ
    fft_signal_c(:,i) = fft(signal(:,i));
end
fft_signal_c = (fft_signal_c/N);
fft_signal_c = fft_signal_c(1:(floor(N/2))+1,:);   % Apenas considerar um lado do↙
espetro
fft_signal = abs(fft_signal_c);                    % fft apenas com o modulo
fft_signal(2:end-1, :) = 2*fft_signal(2:end-1, :); % Multiplicar por 2 todas as↙
frequências, exceto componente DC e a última frequência, para obter espetro unilateral
fft_signal = (fft_signal/sqrt(2)).^2;

f = fs*(1:(N/2))/N; % Array de frequências de 0 a fs/2
f = f(1:(10/df));

fft_signal = fft_signal(1:(10/df), :);
fft_signal_m = sum(fft_signal')/TOTAL_ACQ; % Média de T espetros de potência
fft_signal_mdB = 10*log10(fft_signal_m); % Média em dB

[pico, i(1)]=max(fft_signal_m(2:end)); % Encontar valor maximo do array
i(1) = i(1)+1;
[pico_adj, i(2)] = max(fft_signal_m(i(1)-1:2:i(1)+1)); % Encontrar o máximo entre os↙
vizinhos
i(2)= i(1) - 1*(i(2)==1) + 1*(i(2)==2);
```

```
i = sort(i); % Ordenar as frequências de ordem crescente
fp = [f(i)]; % Obter as frequências correspondentes

% ipdft
% Calcular parâmetros iniciais
omega_a = (2*pi*fp(1))/fs;
omega_b = (2*pi*fp(2))/fs;
UA = real(fft_signal_c(i(1)));
UB = real(fft_signal_c(i(2)));
VA = imag(fft_signal_c(i(1)));
VB = imag(fft_signal_c(i(2)));

% Kopt
Kopt = (((VB-VA)*sin(omega_a))+((UB-UA)*cos(omega_a)))/(UB-UA);
ZA = (VA*((Kopt-cos(omega_a))/(sin(omega_a))))+UA;
ZB = (VB*((Kopt-cos(omega_b))/(sin(omega_b))))+UB;

% Fipdft
Fipdft = (fs/(2*pi))*acos(((ZB*cos(omega_b))-(ZA*cos(omega_a)))/(ZB-ZA));
fprintf("\nFrequencia estimada por IPDFT = %.5f Hz\n", Fipdft);

% Limpar cálculo das frequências
clear pico pico_adj i fp omega_a omega_b UA UB VA VB Kopt ZA ZB fft_signal_c

% Estimar valores médio e eficaz
N_per_period = fs/Fipdft;
N_periods = N/N_per_period;
N_complete_periods = floor(N_periods);
N_avg = round(N_complete_periods * N_per_period);

valor_medio = sum(signal(1:N_avg,1))/N_avg;
fprintf("\nValor medio estimado = %f V\n", valor_medio);

valor_eficaz = sqrt(sum(signal(1:N_avg,1).^2)/N_avg);
fprintf("Valor eficaz estimado = %f V\n", valor_eficaz);

% Limpar valores médio e eficaz
clear N_complete_periods N_avg

% Plot step1
subplot(2,1,2);plot(f,fft_signal_m);

title('Transformada de fourier unilateral do sinal','fontsize',12)
xlabel('f (Hz)')
ylabel('|Potência| (dB W)')
grid on;

signal_centered(:,1) = signal(:,1) - valor_medio;

subplot(2,1,1);
if N_periods > 5
    stem(t(1:round(N_per_period)*5,1), signal_centered(1:round(N_per_period)*5,1));
else
    stem(t, signal, 'filled');
end

xlabel('t (s)')
ylabel('Tensão (V)')
title('Gráfico temporal do sinal','fontsize',12)
```

```
str = sprintf('f = %.2fHz, Valor Médio = %.5fV, Valor Eficaz = %.3fV, N = %d, fs = %.↙
2fHz, Alcance = [-10,10]V',Fipdft, valor_medio, valor_eficaz, N, fs);
sgtitle(str)
```

step6.m

```
clear all
close all

% ---- Load Files with 12V ----
d = uigetdir(pwd, 'Select a folder');
files = dir(fullfile(d, 'x_*_12.mat'));
dfts = [];
freqs = [];
types = [];

% ---- Per file ----
for j = 1:size(files)
    load(strcat(d,'\',files(j).name));
    data = strsplit(files(j).name,'.');
    data = string(data(1));

    % Resoluções e fs reais
    dt = t(2)-t(1);
    fs = 1/dt;
    df = fs/N;

    % DFT
    for i = 1:TOTAL_ACQ
        fft_signal_c(:,i) = fft(signal(:,i));
    end
    fft_signal_c = (fft_signal_c/N);
    fft_signal_c = fft_signal_c(1:(floor(N/2))+1,:);        % Apenas considerar um↙
lado do espetro
    fft_signal = abs(fft_signal_c);                         % fft apenas com o módulo
    fft_signal(2:end-1, :) = 2*fft_signal(2:end-1, :);      % Multiplicar por 2 todas↙
as frequências, menos a componente DC e a última frequencia, que apenas tem uma↙
componente, para obter espetro unilateral
    fft_signal = (fft_signal/sqrt(2)).^2;

    f = fs*(1:(N/2))/N; % Array de frequências de 0 a fs/2
    f = f(1:(10/df));

    fft_signal = fft_signal(1:(10/df), :);
    fft_signal_m = sum(fft_signal')/TOTAL_ACQ; % Média de T espetros de potência
    fft_signal_mdB = 10*log10(fft_signal_m); % Média em dB

    dfts(end+1,:) = fft_signal_m(3:end);
    %dfts(end+1,:) = fft_signal_mdB(3:end);
    types(end+1,:) = data;

    clear dt fft_sign fft_signal_c fft_signal_m
end

figure(1);
subplot(2,1,1);
imagesc(f(3:end), [0 1 2 3 4 5],dfts);
xlabel('Frequency [Hz]')
ylabel('Number of coins')
c = colorbar;
c.Label.String = '|Power| [W]';

% ---- Load Files with 5 'coins' ----
files = dir(fullfile(d, 'x_5_*.mat'));

dfts = [];
```

```
freqs = [];
types = [];

% ---- Per file ----
for j = 1:size(files)
    load(strcat(d,'\',files(j).name));
    data = strsplit(files(j).name,'.');
    data = string(data(1));

    % Resoluções e fs reais
    dt = t(2)-t(1);
    fs = 1/dt;
    df = fs/N;

    % DFT
    for i = 1:TOTAL_ACQ
        fft_signal_c(:,i) = fft(signal(:,i));
    end
    fft_signal_c = (fft_signal_c/N);
    fft_signal_c = fft_signal_c(1:(floor(N/2))+1,:);
    fft_signal = abs(fft_signal_c);
    fft_signal(2:end-1, :) = 2*fft_signal(2:end-1, :);
    fft_signal = (fft_signal/sqrt(2)).^2;

    f = fs*(1:(N/2))/N;
    f = f(1:(10/df));

    fft_signal = fft_signal(1:(10/df), :);
    fft_signal_m = sum(fft_signal')/TOTAL_ACQ;
    fft_signal_mdB = 10*log10(fft_signal_m);

    dfts(end+1,:) = fft_signal_m(3:end);
    %dfts(end+1,:) = fft_signal_mdB(3:end);
    types(end+1,:) = data;
    clear dt fft_sign fft_signal_c fft_signal_m
end

figure(2);
subplot(2,1,1);
imagesc(f(3:end), [4 6 8 9 10 11 12],dfts);
xlabel('Frequency [Hz]')
ylabel('Voltage [V]')
c = colorbar;
c.Label.String = '|Power| [W]';
```

step8_step9.m

```
clear all
close all

% ---- Load all files----
d = uigetdir(pwd, 'Select a folder');
files = dir(fullfile(d, 'x_*.mat'));

data = struct('freq',[],'absValue',[],'coins',[],'voltage',[]);

% ---- Per file ----
for j = 1:size(files)
    load(strcat(d,'\',files(j).name));
    file_name = strsplit(files(j).name,'.');
    file_name = strsplit(string(file_name(1)), '_');
    data.coins(j) = file_name(2);   % in number of coins
    data.voltage(j) = file_name(3); % in volts

    % Resoluções e fs reais
    dt = t(2)-t(1);
    fs = 1/dt;
    df = fs/N;

    % DFT
    for i = 1:TOTAL_ACQ
        fft_signal_c(:,i) = fft(signal(:,i));
    end
    fft_signal_c = (fft_signal_c/N);
    fft_signal_c = fft_signal_c(1:(floor(N/2))+1,:);
    fft_signal = abs(fft_signal_c);
    fft_signal(2:end-1, :) = 2*fft_signal(2:end-1, :);
    fft_signal = (fft_signal/sqrt(2)).^2;

    f = fs*(1:(N/2))/N; % Array de frequencias de 0 a fs/2
    f = f(1:(10/df));

    fft_signal = fft_signal(1:(10/df), :);
    fft_signal_m = sum(fft_signal')/TOTAL_ACQ; % Média de T espetros de potência
    fft_signal_mdB = 10*log10(fft_signal_m);   % Média em dB

    [pico, i(1)]=max(fft_signal_m(2:(4/df)));              % Encontar valor máximo do↙
array
    i(1) = i(1)+1;
    [pico_adj, i(2)] = max(fft_signal_m(i(1)-1:2:i(1)+1)); % Encontrar o máximo entre↙
os vizinhos
    i(2)= i(1) - 1*(i(2)==1) + 1*(i(2)==2);

    i = sort(i);        % Ordenar as frequências de ordem crescente
    fp = [f(i)];        % Obter as frequências correspondentes

    % ipdft
    % Calcular parâmetros iniciais
    omega_a = (2*pi*fp(1))/fs;
    omega_b = (2*pi*fp(2))/fs;
    UA = real(fft_signal_c(i(1)));
    UB = real(fft_signal_c(i(2)));
    VA = imag(fft_signal_c(i(1)));
    VB = imag(fft_signal_c(i(2)));

    % Kopt
    Kopt = (((VB-VA)*sin(omega_a))+((UB-UA)*cos(omega_a)))/(UB-UA);
```

```
    ZA = (VA*((Kopt-cos(omega_a))/(sin(omega_a))))+UA;
    ZB = (VB*((Kopt-cos(omega_b))/(sin(omega_b))))+UB;

    % Fipdft
    Fipdft = (fs/(2*pi))*acos(((ZB*cos(omega_b))-(ZA*cos(omega_a)))/(ZB-ZA));
    fprintf("\nNmr moedas: %d\nValor absoluto pico: %f\nFrequência estimada por IPDFT:↙
%.5f Hz\n", data.coins(j), pico, Fipdft);

    data.freq(j) = Fipdft;    % in Hz
    data.absValue(j) = pico;  % in volts

    clear dt fft_sign fft_signal_c fft_signal_m

end

%|CP|=f(CP,m)
p = polyfitn([data.freq', data.coins'],data.absValue',3);
fprintf("\nRegression |CP|=f(CP,m):\n");
display(polyn2sympoly(p))

%CP=f(|CP|,m)
p1 = polyfitn([data.absValue', data.coins'],data.freq',3);
fprintf("\nRegression CP=f(|CP|,m):\n");
display(polyn2sympoly(p1))

%m=f(CP,|CP|)
Polyfit_aprox = polyfitn([data.freq', data.absValue'],data.coins',3);
fprintf("\nRegression m=f(CP,|CP|):\n");
display(polyn2sympoly(Polyfit_aprox))

[xg,yg]=meshgrid(0:0.5:5);
zg = polyvaln(p,[xg(:),yg(:)]);
surf(xg,yg,reshape(zg,size(xg)))
hold on
plot3(data.freq', data.coins',data.absValue', 'o','Color','k','MarkerSize', 5,↙
'MarkerFaceColor','k');

title('3D surface plot (step 8)');
xlabel('CP [Hz]');
zlabel('|CP| [W]');
ylabel('Number of Coins');
hold off

clearvars -except p p1 Polyfit_aprox
```

step10.m

```
clear fft_signal_c

fs = 20e3;       % Acquisition rate
TOTAL_ACQ = 3;   % Number of repetitions
N = 400e3;       % Number of acquisitions

% -----visualização do DAQ-----
dispositivo = daqlist;
placa_ID = dispositivo.DeviceID;
placa_vendedor = dispositivo.VendorID;
deviceInfo = dispositivo{1, "DeviceInfo"};

% -----Programação do DAQ-----
dq = daq(placa_vendedor);
dq.Rate = fs;
ch = addinput(dq, placa_ID, 'ai0', 'Voltage');
ch.Range=[-2,2];
fs = dq.Rate;

% ---- Carregar buffer ----
start(dq, "Continuous")
fprintf("[INFO] A carregar buffer...\n")
for i=1:TOTAL_ACQ
    [buffer(:, i), t(:, i)] = read(dq, N, "OutputFormat", "Matrix");
    fprintf("    > [INFO] Foram obtidas %d amostras do sinal [%d/%d].\n", N, i,↙
TOTAL_ACQ);
end
stop(dq)
fprintf("[INFO] Buffer carregado com sucesso!\n")

% ---- Main Loop ----
DlgH = figure('Name','Step 10','NumberTitle','off');
H = uicontrol('Style', 'PushButton', ...
                    'String', 'Stop', ...
                    'Callback', 'delete(gcbf)');
fig_annotation = annotation('textbox',[.7 .5 .3 .3],'String','','FitBoxToText','on');

cur_index = 1;
while (ishandle(H))
    % ---- Estimar CP e obter |CP|----
    % Resoluções e fs reais
    dt = t(2)-t(1);
    fs = 1/dt;
    df = fs/N;

    % DFT
    for i = 1:TOTAL_ACQ
        fft_signal_c(:,i) = fft(buffer(:,i));
    end
    fft_signal_c = (fft_signal_c/N);
    fft_signal_c = fft_signal_c(1:(floor(N/2))+1,:);        % Apenas considerar um↙
lado do espetro
    fft_signal = abs(fft_signal_c);                         % fft apenas com o módulo
    fft_signal(2:end-1, :) = 2*fft_signal(2:end-1, :);      % Multiplicar por 2 todas↙
as frequências menos a componente DC e a ultima frequência, que apenas tem uma↙
componente, para obter espetro unilateral
    fft_signal = (fft_signal/sqrt(2)).^2;

    f = fs*(1:(N/2))/N; % Array de frequências de 0 a fs/2
    f = f(1:(10/df));
```

```
    fft_signal = fft_signal(1:(10/df), :);
    fft_signal_m = sum(fft_signal')/TOTAL_ACQ;              % Média de T espetros de↙
potência
    fft_signal_mdB = 10*log10(fft_signal_m);                % Média em dB

    [pico, index_pico]=max(fft_signal_m(2:(4/df)));         % Encontar valor máximo do↙
array
    i(1) = index_pico+1;
    [pico_adj, i(2)] = max(fft_signal_m(i(1)-1:2:i(1)+1));  % Encontrar o máximo entre↙
os vizinhos
    i(2)= i(1) - 1*(i(2)==1) + 1*(i(2)==2);

    i = sort(i);        % Ordenar as frequências de ordem crescente
    fp = [f(i)];        % Obter as frequências correspondentes

    % ipdft
    % Calcular parâmetros iniciais
    omega_a = (2*pi*fp(1))/fs;
    omega_b = (2*pi*fp(2))/fs;
    UA = real(fft_signal_c(i(1)));
    UB = real(fft_signal_c(i(2)));
    VA = imag(fft_signal_c(i(1)));
    VB = imag(fft_signal_c(i(2)));

    % Kopt
    Kopt = (((VB-VA)*sin(omega_a))+((UB-UA)*cos(omega_a)))/(UB-UA);
    ZA = (VA*((Kopt-cos(omega_a))/(sin(omega_a))))+UA;
    ZB = (VB*((Kopt-cos(omega_b))/(sin(omega_b))))+UB;

    % Fipdft
    Fipdft = (fs/(2*pi))*acos(((ZB*cos(omega_b))-(ZA*cos(omega_a)))/(ZB-ZA));

    % ---- Display dos resultados ----
    plot(f,fft_signal_mdB)
    xlabel("Freq [Hz]")
    ylabel("|Power [dB W]|")
    title("DFT and real time predictions")
    xline(Fipdft,'-',{'Fipdft','CP'});
    text(index_pico*df,10*log10(pico),'\leftarrow |CP|')
    %set(fig_annotation,'String',sprintf("Unbalacing weight: %.3f g\nRotation speed:↙
%.1f RPM", polyvaln(Polyfit_aprox,[Fipdft,sqrt(sum(fft_signal_m(index_pico-2:↙
index_pico+2).^2))])*10, Fipdft*60))
    set(fig_annotation,'String',sprintf("Unbalacing weight: %.3f g\nRotation speed: %.↙
1f RPM", polyvaln(Polyfit_aprox,[Fipdft,pico])*10, Fipdft*60))
    drawnow

    % ---- Adquirir  mais dados e substituir no buffer ----
    start(dq, "Continuous")
    [buffer(:, cur_index), t(:, cur_index)] = read(dq, N, "OutputFormat", "Matrix");
    stop(dq)
    fprintf("[INFO] Foram obtidas %d amostras do sinal\n", N)

    cur_index=mod(cur_index+1,TOTAL_ACQ)+1;

    clear fft_signal_c
end
```

step1.m

```
clear fft_signal_c

fs = 20e3;      % Acquisition rate
TOTAL_ACQ = 3; % Process repeated 3 times, average computed
N = 1000e3;     % Number of acquisitions

% ----- Visualização do DAQ -----
dispositivo = daqlist;
placa_ID = dispositivo.DeviceID;
placa_vendedor = dispositivo.VendorID;
deviceInfo = dispositivo{1, "DeviceInfo"};

% ----- Programação do DAQ -----
dq = daq(placa_vendedor);
dq.Rate = fs;
ch = addinput(dq, placa_ID, 'ai0', 'Voltage');
ch.Range=[-2,2];

% ---- Carregar buffer ----
start(dq, "Continuous")
fprintf("[INFO] A carregar buffer...\n")
for i=1:TOTAL_ACQ
    [buffer(:, i), t(:, i)] = read(dq, N, "OutputFormat", "Matrix");
    fprintf("    > [INFO] Foram obtidas %d amostras do sinal [%d/%d].\n", N, i,↙
TOTAL_ACQ);
end
stop(dq)
fprintf("[INFO] Buffer carregado com sucesso!\n")

% ---- Main Loop ----
cur_index = 1;
while (1)
    % ---- Estimar CP e obter |CP| ----
    % Resoluções e fs reais
    dt = t(2)-t(1);
    fs = 1/dt;
    df = fs/N;

    % DFT
    for i = 1:TOTAL_ACQ
        fft_signal_c(:,i) = fft(buffer(:,i));
    end
    fft_signal_c = (fft_signal_c/N);
    fft_signal_c = fft_signal_c(1:(floor(N/2))+1,:);    % Apenas considerar um lado do↙
espetro
    fft_signal = abs(fft_signal_c);                     % fft apenas com o módulo
    fft_signal(2:end-1, :) = 2*fft_signal(2:end-1, :);  % Multiplicar por 2 todas as↙
frequências, menos a componente DC e a última frequência, que apenas tem uma↙
componente, para obter espetro unilateral
    fft_signal = (fft_signal/sqrt(2)).^2;

    f = fs*(1:(N/2))/N; % Array de frequências de 0 a fs/2
    f = f(1:(10/df));

    fft_signal = fft_signal(1:(10/df), :);
    fft_signal_m = sum(fft_signal')/TOTAL_ACQ; % Média de T espetros de potência
    fft_signal_mdB = 10*log10(fft_signal_m);   % Média em dB

    [pico, index_pico]=max(fft_signal_m(2:(4/df)));        % Encontar valor máximo do↙
array
```

```
    i(1) = index_pico+1;
    [pico_adj, i(2)] = max(fft_signal_m(i(1)-1:2:i(1)+1)); % Encontrar o máximo entre↙
os vizinhos
    i(2)= i(1) - 1*(i(2)==1) + 1*(i(2)==2);

    i = sort(i); % Ordenar as frequências de ordem crescente
    fp = [f(i)]; % Obter as frequências correspondentes

    % ipdft
    % Calcular parâmetros iniciais
    omega_a = (2*pi*fp(1))/fs;
    omega_b = (2*pi*fp(2))/fs;
    UA = real(fft_signal_c(i(1)));
    UB = real(fft_signal_c(i(2)));
    VA = imag(fft_signal_c(i(1)));
    VB = imag(fft_signal_c(i(2)));

    % Kopt
    Kopt = (((VB-VA)*sin(omega_a))+((UB-UA)*cos(omega_a)))/(UB-UA);
    ZA = (VA*((Kopt-cos(omega_a))/(sin(omega_a))))+UA;
    ZB = (VB*((Kopt-cos(omega_b))/(sin(omega_b))))+UB;

    % Fipdft
    Fipdft = (fs/(2*pi))*acos(((ZB*cos(omega_b))-(ZA*cos(omega_a)))/(ZB-ZA));

    % ---- Enviar dados ----
    thingSpeakWrite(2004335,'Fields',[1,2],'Values',{Fipdft,↙
pico},'WriteKey','KIVCVCLU824Y827Q')

    % ---- Adquirir  mais dados e substituir no buffer ----
    start(dq, "Continuous")
    [buffer(:, cur_index), t(:, cur_index)] = read(dq, N, "OutputFormat", "Matrix");
    stop(dq)
    fprintf("[INFO] Foram obtidas %d amostras do sinal\n", N)

    cur_index=mod(cur_index+1,TOTAL_ACQ)+1
    clear fft_signal_c
end
```

step2.m

```
clear fft_signal_c

% ---- MQTT Credentials ----
clientId = 'NgsoCS0AIjwZMA4XNwgPIwM';
username = 'NgsoCS0AIjwZMA4XNwgPIwM';
password = 'JP+Xfilvo02qkC5dmJM9XwtI';

brokerAddress = "tcp://mqtt3.thingspeak.com";
topicFreq = "channels/2004335/publish/fields/field1";
topicAmp = "channels/2004335/publish/fields/field2";
port = 1883;
javaaddpath('matlab_mqtt_source\mqttasync.jar');
javaaddpath('matlab_mqtt_source\jar\org.eclipse.paho.client.mqttv3-1.1.0.jar');
addpath('matlab_mqtt_source');

mqClient = mqtt(brokerAddress,'Port', port, 'Username', username, 'Password',↙
password,'ClientID', clientId);

fs = 20e3;     % Acquisition rate
TOTAL_ACQ = 3; % Number of acquisitions
N = 1000e3;    % Number of acquisitions

% ----- Visualização do DAQ -----
dispositivo = daqlist;
placa_ID = dispositivo.DeviceID;
placa_vendedor = dispositivo.VendorID;
deviceInfo = dispositivo{1, "DeviceInfo"};

% ----- Programação do DAQ -----
dq = daq(placa_vendedor);
dq.Rate = fs;
ch = addinput(dq, placa_ID, 'ai0', 'Voltage');
ch.Range=[-2,2];

% ---- Carregar buffer ----
start(dq, "Continuous")
fprintf("[INFO] A carregar buffer...\n")
for i=1:TOTAL_ACQ
    [buffer(:, i), t(:, i)] = read(dq, N, "OutputFormat", "Matrix");
    fprintf("    > [INFO] Foram obtidas %d amostras do sinal [%d/%d].\n", N, i,↙
TOTAL_ACQ);
end
stop(dq)
fprintf("[INFO] Buffer carregado com sucesso!\n")

% ---- Main Loop ----
cur_index = 1;
while (1)
    % ---- Estimar CP e obter |CP|----
    % Resoluções e fs reais
    dt = t(2)-t(1);
    fs = 1/dt;
    df = fs/N;

    % DFT
    for i = 1:TOTAL_ACQ
        fft_signal_c(:,i) = fft(buffer(:,i));
    end
    fft_signal_c = (fft_signal_c/N);
    fft_signal_c = fft_signal_c(1:(floor(N/2))+1,:);   % Apenas considerar um lado do↙
```

```
espetro
    fft_signal = abs(fft_signal_c);                    % fft apenas com o módulo
    fft_signal(2:end-1, :) = 2*fft_signal(2:end-1, :); % Multiplicar por 2 todas as↙
frequências, menos a componente DC e a última frequência, que apenas tem uma↙
componente, para obter espetro unilateral
    fft_signal = (fft_signal/sqrt(2)).^2;

    f = fs*(1:(N/2))/N; % Array de frequências de 0 a fs/2
    f = f(1:(10/df));

    fft_signal = fft_signal(1:(10/df), :);
    fft_signal_m = sum(fft_signal')/TOTAL_ACQ; % Média de T espetros de potência
    fft_signal_mdB = 10*log10(fft_signal_m);   % Média em dB

    [pico, index_pico]=max(fft_signal_m(2:(4/df)));        % Encontar valor máximo do↙
array
    i(1) = index_pico+1;
    [pico_adj, i(2)] = max(fft_signal_m(i(1)-1:2:i(1)+1)); % Encontrar o máximo entre↙
os vizinhos
    i(2)= i(1) - 1*(i(2)==1) + 1*(i(2)==2);

    i = sort(i);  % Ordenar as frequências de ordem crescente
    fp = [f(i)];  % Obter as frequências correspondentes

    % ipdft
    % Calcular parâmetros iniciais
    omega_a = (2*pi*fp(1))/fs;
    omega_b = (2*pi*fp(2))/fs;
    UA = real(fft_signal_c(i(1)));
    UB = real(fft_signal_c(i(2)));
    VA = imag(fft_signal_c(i(1)));
    VB = imag(fft_signal_c(i(2)));

    % Kopt
    Kopt = (((VB-VA)*sin(omega_a))+((UB-UA)*cos(omega_a)))/(UB-UA);
    ZA = (VA*((Kopt-cos(omega_a))/(sin(omega_a))))+UA;
    ZB = (VB*((Kopt-cos(omega_b))/(sin(omega_b))))+UB;

    % Fipdft
    Fipdft = (fs/(2*pi))*acos(((ZB*cos(omega_b))-(ZA*cos(omega_a)))/(ZB-ZA));

    % ---- Enviar dados ----
    formatSpec = '%.20f';
    publish(mqClient, topicFreq, num2str(Fipdft));
    publish(mqClient, topicAmp, num2str(pico, formatSpec));

    % ---- Adquirir mais dados e substituir no buffer ----
    start(dq, "Continuous")
    [buffer(:, cur_index), t(:, cur_index)] = read(dq, N, "OutputFormat", "Matrix");
    stop(dq)
    fprintf("[INFO] Foram obtidas %d amostras do sinal\n", N)

    cur_index=mod(cur_index+1,TOTAL_ACQ)+1
    clear fft_signal_c
end
```

step3.m

```
clear fft_signal_c

% ---- API configuration ----
thingSpeakURL = 'http://api.thingspeak.com/';
thingSpeakWriteURL = [thingSpeakURL 'update.json'];
writeApiKey = 'KIVCVCLU824Y827Q';

addpath('matlab_json_source\jsonlab');

fs = 20e3;     % Acquisition rate
TOTAL_ACQ = 3; % Number of acquisitions
N = 1000e3;    % Number of acquisitions

% ----- Visualização do DAQ -----
dispositivo = daqlist;
placa_ID = dispositivo.DeviceID;
placa_vendedor = dispositivo.VendorID;
deviceInfo = dispositivo{1, "DeviceInfo"};

% ----- Programação do DAQ -----
dq = daq(placa_vendedor);
dq.Rate = fs;
ch = addinput(dq, placa_ID, 'ai0', 'Voltage');
ch.Range=[-2,2];

% ---- Carregar buffer ----
start(dq, "Continuous")
fprintf("[INFO] A carregar buffer...\n")
for i=1:TOTAL_ACQ
    [buffer(:, i), t(:, i)] = read(dq, N, "OutputFormat", "Matrix");
    fprintf("    > [INFO] Foram obtidas %d amostras do sinal [%d/%d].\n", N, i,↙
TOTAL_ACQ);
end
stop(dq)
fprintf("[INFO] Buffer carregado com sucesso!\n")

% ---- Main Loop ----
cur_index = 1;
while (1)
    % ---- Estimar CP e obter |CP|----
    % Resoluções e fs reais
    dt = t(2)-t(1);
    fs = 1/dt;
    df = fs/N;

    % DFT
    for i = 1:TOTAL_ACQ
        fft_signal_c(:,i) = fft(buffer(:,i));
    end
    fft_signal_c = (fft_signal_c/N);
    fft_signal_c = fft_signal_c(1:(floor(N/2))+1,:);   % Apenas considerar um lado do↙
espetro
    fft_signal = abs(fft_signal_c);                    % fft apenas com o módulo
    fft_signal(2:end-1, :) = 2*fft_signal(2:end-1, :); % Multiplicar por 2 todas as↙
frequências, menos a componente DC e a última frequência, que apenas tem uma↙
componente, para obter espetro unilateral
    fft_signal = (fft_signal/sqrt(2)).^2;

    f = fs*(1:(N/2))/N; % Array de frequências de 0 a fs/2
    f = f(1:(10/df));
```

```
    fft_signal = fft_signal(1:(10/df), :);
    fft_signal_m = sum(fft_signal')/TOTAL_ACQ; % Média de T espetros de potência
    fft_signal_mdB = 10*log10(fft_signal_m);   % Média em dB

    [pico, index_pico]=max(fft_signal_m(2:(4/df)));        % Encontar valor máximo do↙
array
    i(1) = index_pico+1;
    [pico_adj, i(2)] = max(fft_signal_m(i(1)-1:2:i(1)+1)); % Encontrar o máximo entre↙
os vizinhos
    i(2)= i(1) - 1*(i(2)==1) + 1*(i(2)==2);

    i = sort(i); % Ordenar as frequências de ordem crescente
    fp = [f(i)]; % Obter as frequências correspondentes

    % ipdft
    % Calcular parâmetros iniciais
    omega_a = (2*pi*fp(1))/fs;
    omega_b = (2*pi*fp(2))/fs;
    UA = real(fft_signal_c(i(1)));
    UB = real(fft_signal_c(i(2)));
    VA = imag(fft_signal_c(i(1)));
    VB = imag(fft_signal_c(i(2)));

    % Kopt
    Kopt = (((VB-VA)*sin(omega_a))+((UB-UA)*cos(omega_a)))/(UB-UA);
    ZA = (VA*((Kopt-cos(omega_a))/(sin(omega_a))))+UA;
    ZB = (VB*((Kopt-cos(omega_b))/(sin(omega_b))))+UB;

    % Fipdft
    Fipdft = (fs/(2*pi))*acos(((ZB*cos(omega_b))-(ZA*cos(omega_a)))/(ZB-ZA));

    % ---- Enviar dados de maneira alternativa ----
    payload=struct('api_key', writeApiKey, 'field1', Fipdft, 'field2', pico);

    %response = webwrite(thingSpeakWriteURL,payload, weboptions('MediaType',↙
'application/json'))

    % ---- Enviar dados em JSON formatted payload e mostrar size ----

    data2json.api_key = writeApiKey;
    data2json.field1 = Fipdft;
    data2json.field2 = pico;
    payload_json = savejson('',data2json)
    disp(size(payload_json))

    response = webwrite(thingSpeakWriteURL, payload_json, weboptions('MediaType',↙
'application/json'))

    % ---- Enviar dados em MSGPACK formatted payload e mostrar size ----

    data2msg=struct('api_key', writeApiKey, 'field1', Fipdft, 'field2', pico);
    payload_msg = savemsgpack('',data2msg)
    disp(size(payload_msg))

    % No readability with MSGPACK:
    % response=webwrite(thingSpeakWriteURL, payload_msg, weboptions('MediaType',↙
'application/json'))

    % ---- Adquirir  mais dados e substituir no buffer ----
```

```
    start(dq, "Continuous")
    [buffer(:, cur_index), t(:, cur_index)] = read(dq, N, "OutputFormat", "Matrix");
    stop(dq)
    fprintf("[INFO] Foram obtidas %d amostras do sinal\n", N)

    cur_index=mod(cur_index+1,TOTAL_ACQ)+1
    clear fft_signal_c
end
```