Abstract

A simple and inexpensive system capable of displaying relevant power quality data of a residential voltage signal is presented, and it is based on Arduino, Python and Kivy. A circuit for adjusting the voltage signal for sampling is suggested. The software of the system allows the transmission of data with any device capable of serial communication at a 2000000 Baud rate, although an Arduino NANO/UNO board is recommended. A board based on an Arduino NANO R3 was used. The software plots the last 6 cycles contained in the data by default, but lets analyze any amount of cycles. It provides the RMS voltage, the peak voltage and the total harmonic distortion of the scrutinized cycles, up to the 25th harmonic. The software is also capable of calculating the fast Fourier transform of the studied cycles, and returning the corresponding plot. A sample of data captured at Maracaibo, Venezuela is shown.

Key words: Power quality; monitor; Venezuela; Arduino; Python.

1 Introduction

The current state of Venezuela’s electrical system is extremely dire, due to a long and ongoing period of mismanagement [1]. All the monitoring, maintenance and expansion that is required to sustain the system hasn’t been done in a considerable amount of years. The consequences of the mismanagement can be summed up as the decay of the system, which has brought many power disturbances such as voltage sags and surges, being these two disturbances a prominent source for damage of electrical appliances.

For the past few years, no data on the topic has been publicly disclosed by official sources. Although it is agreed among many Venezuelans that the electric service is in bad shape, it is extremely hard to make an objective analysis of its current state without using any sort of data. Taking into account that there’s no available information on the subject, the only possible way to get data about Venezuela’s electrical system is to use power quality monitoring equipment, but being that Venezuela’s economy is in an appalling state, this option turns out to be extremely expensive for locals. Gathering data city-wide is desirable, but making a full scale analysis of all the distribution circuits of an entire city would require a significant amount of equipment.

Thanks to the rise of micro-controllers in the modern era, it is possible to access low-cost devices capable of doing analog to digital conversions, which are required for gathering the data of the electricity. A simple and inexpensive system that works as a solution to this problem is showcased in this paper. Said system is able to continuously sample, measure the RMS value of the voltage, the peak voltage, and the total harmonic distortion (THD) considering up to the 25th harmonic.

The system is comprised by two parts: the data gathering part, which uses its own piece of software and an Arduino board alongside a circuit; and the data analysis part, which processes and displays in a computer all the gathered data. To achieve a low cost approach, an Arduino-based board is used alongside a few spare parts for a circuit, and a Dell Vostro 3500 with Linux, Python and Kivy. The reasoning behind these selections is that they represent free software paired with very low cost hardware. This combination is able to capture, process and analyze data from a residential voltage signal, which in Venezuela’s case is 120 VRMS at 60 Hz [2]. The overall goal of this system is to make a scalable solution that could be used for making an analysis of the status of all the distribution circuits of Maracaibo, Venezuela.

2 Data Gathering

For the design of the system a board based on the Arduino NANO R3 was used, due to its low cost. Under this setup an Arduino NANO or UNO can be used, or just an Atmega 328P for that matter. Other MCU devices can be used, but that will deviate the system from what is presented in this paper.
It was established that the workload of the Arduino board should be kept at a minimum, and that a computer must be used to do the signal processing. This decision was not only based on the low processing power of the Atmega 328P, but in the fact that increased CPU activity of the micro-controller leads to generation of noise [3], which is undesirable. There were a lot measures taken for ensuring the lowest amount of noise possible (excluding shielding the device), and they comprise the following:

- Setting all unused pins as inputs.
- Connecting all unused pins to ground.
- Utilizing the noise reduction mode for doing analog readings.
- Disabling the digital input buffer of analog pins.
- Using decoupling capacitors for the analog reference and power pins.
- Not using target computer as source of power.
- Keeping analog and power lines as close as possible.

These measures were based on the fact that floating pins generate noise [4], and on various EMC, grounding and power electronics concepts [5],[6],[7],[8]. Any unused components of the Atmega 328P were disabled. Although the following wasn’t achieved for this system, isolating the Arduino board as much as possible from the target computer would reduce the electrical noise sent to the board by the computer. The best method of isolation is avoiding any electrical connections whatsoever.

To stay in compliance with the Nyquist sampling theorem [9], a 3600 Hz sampling rate was chosen. This sampling rate not only lets capture the data of the fundamental frequency of the residential voltage signal, which is 60 Hz, but it also lets capture data for all harmonics of said signal up to the 25th one.

For the presented scenario the Nyquist frequency is 1500 Hz, which means that a 3000 Hz or higher sampling rate is required. Although increasing various times the sampling rate is possible and would in theory provide higher quality data, that would also require increasing the clock frequency of the analog to digital converter (ADC), which would lead to reducing the resolution of the ADC. The default frequency of the ADC clock for the Arduino board is 125 kHz. Any value above 200 kHz makes 10 bits readings unreliable, forcing to use 8 bits-only readings [3]. Due to the way the prescalers of the ADC and the system clock frequency work, it is not possible to set the ADC clock frequency to any value between 125 kHz and 200 kHz.

Between the limitations of the ADC of the Atmega 328P, lies the input restriction of 0 - 5 V. This implies that signal manipulation is required in order to sample the residential voltage. Considering that potential transformers are a source of noise and could ramp up the overall cost of the system, the use of a voltage divider with resistors was preferred and is recommended. This is acceptable due to the internal resistance of the ADC being 100MΩ[3], being that a set of values that are significantly lower can be used. Alongside the voltage divider, an external DC source of 3.3 V was used to provide an offset to the signal, leaving the following equivalent circuit:

\[ \frac{V_o}{V_i} = \frac{5\,\text{kΩ}}{1\,\text{MΩ} + 5\,\text{kΩ}} \approx 0.0050 \quad (1) \]

Under this configuration the 5 kΩ resistor perceives approximately 0.5% of the residential voltage at any time, but it is to be taken into account that this value may vary if the resistors experiment temperature changes [4]. This implies that for a 120 VRMS signal there’s an output of 0.6 VRMS with the same properties as the input. Said output has a set of peak values of ±0.6 × √2 V, which transgresses the input limit of the ADC, being that there’s no tolerance for negative values. Due to this fact it was decided to use a 3.3 V source for offsetting the output signal from the voltage divider, making the peak values of the resulting signal 3.3 V ±0.6 × √2 V, which are both positive values under 5 V. This configuration also provides great tolerance against voltage surges, being that an increase of up to 100% in amplitude would still produce an upper peak under 5 V, and a voltage surge of 100% or higher is never to be expected in Maracaibo [10]. The voltage provided by the 3.3 V DC source was filtered with a decoupling capacitor.

It is worth noting that although modern regulations establish that for the THD calculation all the harmonics up to the 50th should be considered [11], the aim of this device isn’t being equal to professional power quality equipment, the main goal is to give insight about the current state of Venezuela’s electric service based on Maracaibo’s data, as there is no available information at a research level. It is possible to set the Atmega 328P to sample at the 6000 Hz required for registering up to the 50th harmonic without modifying the prescalers,
but that would require scrapping the use of the Arduino
C/C++ functions, and take an approach directly fo-
cused in plain C or Assembly. This approach is planned
to be added in the future.

The code required for sampling the resulting signal and
then sending it to the target computer is very short and
simple. A snippet of the code that demonstrates the
whole process from a high level perspective, can be seen
here:

```c
20 void setup() {
21 setAllPinsAsInput(); // Reduces noise
22 analogRead(A2); // Set ADMUX, initial read
23 deactivateDigitalInputs(); // Reduces noise
24 Serial.begin(2000000); }
25 void loop() {
26 StartTime = micros();
27 value = NRADC(); // ADC with noise reduction
28 sprintf(buffer, "%d", value);
29 Serial.println(buffer);
30 while (micros() < StartTime + 162); }
```

The entire code only has 48 lines, and is represented
overall by the prior snippet. The use of a buffer as op-
posed to a String data type is due to preventing mem-
ory allocation issues, which is a prominent problem with
C/C++ coding in general [12]. Using the String data
type in the dynamic memory of a micro-controller can
lead to program crashing under continuous use, due to
memory fragmentation.

The data is sent with a Baud rate of 2000000, which
isn’t strictly necessary but it ensures that each reading
takes under approximately 277 µs to be sent, and there-
fore the sampling rate is very close to 3600 Hz. The use
of an Arduino board for gathering and sending the data
isn’t required, but it opens up the option of making a low
cost Internet of Things infrastructure [13], which could
be used for sending to the cloud the data of all the dis-
tribution circuits of an entire city, such as Maracaibo, in
order to make a deep analysis.

3 Data Processing

For this section, it was decided to work with Python and
Kivy. The Matplotlib, NumPy and SciPy modules are
able to handle all the operations that the system is re-
quired to make. The NumPy module is only used as an
auxiliary tool for plotting with Matplotlib and making
operations related to the THD. The SciPy module is ex-
clusive for making the fast Fourier transform (FFT),
which is used both for calculating the THD and plotting
the FFT of the last 6 cycles of the sampled voltage sig-
nal. The PySerial module is used to establish the serial
connection and receive each line of data. It is compatibi-
```python
    def getVRMS(values):
        total = 0
        VRMS = (total/(len(values)))**(1/2)
        return VRMS
    def getVP(values):
        return max(max(values),abs(min(values)))

The expression utilized for the values of the FFT, which
is used for plotting and getting the THD, is the following:

\[
\hat{V}[k] = \sum_{n=0}^{3599} e^{-2\pi j \frac{kn}{3600}} V[n]
\] (3)

For this expression both \( V \) and \( \hat{V} \) have 3600 values,
which represents the amount of values within 6 cycles

```
for a sampling rate of 3600 Hz. The FFT algorithm of SciPy returns all the positive values in the first half [15] of $\hat{V}$, while it returns all negative values in the remaining half. For both plotting the FFT and calculating of the THD, only the modulus of the values of the transform are used. The FFT returned by SciPy under this scenario covers up to 1800 Hz, which is over the 1500 Hz of the 25th harmonic. The expression used for calculating the THD is the following:

$$THD = \sqrt{\sum_{n=2}^{25} \frac{|\hat{V}_n|^2}{|\hat{V}_1|}}$$

(4)

Where $\hat{V}_n$ represents the voltage of each n harmonic in the frequency domain, and $\hat{V}_1$ represents the voltage of the fundamental harmonic of the signal.

It is important to consider that the FFT algorithm of SciPy always truncates the input signal [15]. Said truncation can be modeled as the multiplication of a corresponding infinite signal with a window function. This translates as a convolution in the frequency domain [9], which leads into spectral leakage. This effect inherently introduces a deviation in the THD calculation, being that there’s going to be values above zero for all frequencies in $\hat{V}$. Nonetheless, the behavior of the spectral leakage for an ideal discrete sine wave can be used to identify whether or not there’s a presence of harmonics in the sampled signal. The FFT of a discrete sine wave of 60 Hz, 6 cycles and 60 values per cycle can be seen in the following figure:

As can be observed in figure 2, after the expected spike at 60 Hz the values started to drop. This means that for there to be energy transmission at any other positive frequency, there has to be another spike that will break and reintroduce the trend of decaying values. By adding a similar discrete sine wave of amplitude $\frac{1}{2}$ and 180 Hz to the original sine wave, we get the following behavior:

As previously asserted, the introduction of a harmonic broke the lowering pattern due to a spike, and after said spike the pattern returned. Due to this fact alone, it was decided only to take into account the voltages of harmonic frequencies that were higher than the prior and following value, in such way that only spikes can be considered for the calculation. This negates the values on any harmonic frequencies that don’t meet this criteria, and therefore their contribution to the calculation of the THD is none.

In the following figure can be seen the GUI for establishing connection with the target PC, and start gathering data:

This GUI is its own piece of software, and it is designed to be simple, intuitive and consume the least amount of resources possible. Once the connection is established, the Kivy environment that runs the program freezes and only the data storing is left working in the background.
In figure 5 can be seen the connection screen: This system has a second GUI, which serves as the main one and it is where all the data gets displayed. The main GUI has its own code that processes the stored data. The main GUI can be seen in action in figure 6:

The sample gets plotted using Matplotlib, and the data displayed on the right side is calculated based on the values used for plotting. The frequency displayed is only referential and not actually calculated, although this is planned to be added in the future. The main GUI can work as long as there’s stored data, regardless there’s transmission happening or not. The ability to analyze and make FFT-related operations to any range of cycles is also planned to be added in the future.

4 Analysis of Sample

The last 6 cycles of a sample taken on Maracaibo, Venezuela on May 6th of 2020, at 4:01 P.M. are in the following figure:

As can be seen in the figure 7, there’s distortion in the peaks of the sine wave. The distortion isn’t present on the linear portions of the sine wave, therefore there’s no clear indication that the peak distortion is due to noise. The distortion isn’t really affecting the RMS value, being that it is hovering around 120 VRMS, as it should according to Venezuela’s national electric code [2]. The overall THD of the sample indicates that there’s definitely presence of harmonics, but they aren’t prominent and they may be conditioned by noise. In figure 8 the FFT of the sample can be seen:

The vast majority of the energy is being transmitted at 60 Hz, but there’s energy transmission occurring at practically all harmonics of the fundamental frequency. This scenario opens up the possibility that the readings are being affected by noise, being that there’s no other prevalent spike in the spectrum. A comparison with a close to an ideal sine wave signal has to be drawn using the same hardware, to determine whether or not noise is
having a great impact. Also, a comparison with professional power quality equipment would be ideal to confirm whether the real THD value is close or not to what is being calculated by the system.

5 Conclusions

The system is capable of perceiving and storing the value changes that come with voltage sags and surges, so it can be used for their study. It requires inexpensive hardware and takes advantage of free software. It can be easily scaled to analyze Maracaibo’s electrical system, having only as a setback the poor state of Internet connectivity in the city. Comparison with professional power quality equipment and with a sine wave from a signal generator is pending. It has great potential for expanding functionality.

References

[1] R. Dube, “Venezuela’s Power Grid Afflicted by Brain Drain, Corruption”. The Wall Street Journal, 2019.
[2] ——, National Electric Code FONDONORMA 200:2004. CODELECTRA, 2004.
[3] ——, megaAVR Data Sheet Rev. A. Microchip Technology Inc., 2018.
[4] P. Wilson, The Circuit Designer’s Companion. Newnes, 2013.
[5] R. Morrison, Grounding and Shielding Techniques in Instrumentation. Wiley-Interscience, 1986.
[6] J. Goedbloed, Electromagnetic Compatibility. Prentice Hall, 1992.
[7] J. Kraus, Electromagnetics. McGraw Hill, 1991.
[8] N. Ross, The Essence of Power Electronics. Prentice Hall, 1997.
[9] S. Smith, The Scientist and Engineer’s Guide to Digital Signal Processing. California Technical Publishing, 1999.
[10] A. Gómez and G. Puche, “Voltage surges in the Venezuelan occidental electrical system taking into account the fault-induced delayed voltage recovery (FIDVR) phenomenon: A Case Study”. Universidad Rafael Urdaneta, 2014.
[11] ——, IEEE Standard 519-2014. IEEE, 2014.
[12] B. Powers, D. Trench, E. Berger and A. McGregor, “Mesh: Compact Memory Management for C/C++ Applications”. arXiv preprint [arXiv:1902.04738] 2019.
[13] V. Nundloll, B. Porter, G. Blair, J. Cosby, B. Emmett, B. Winterboum, G. Dean, P. Beattie, R. Shaw, D. Jones, D. Chadwick, M. Brown, W. Shelley and I. Ullah, “The Design and Deployment of an End-to-end IoT Infrastructure for the Natural Environment”. arXiv preprint [arXiv:1901.06270], 2019.
[14] ——, “Classes”, in PySerial API, PySerial, 2019.
[15] ——, “Fourier Transforms (scipy.fft)”, in SciPy v1.4.1 Reference Guide, SciPy, 2019.