JME
Journal of Mechatronics Engineering

Versatile IoT system for Cloud-based sensor monitoring

Ayrton A. M. Silva[1], Sandro C. S. Jucá[1],
Leonardo S. Costa[1], Paulo M. M. Silva[1],
Renata I. S. Pereira[2]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE), Brazil
[2]Universidade Federal do Ceará (UFC), Brazil

<ayrton.alexmds@gmail.com>, <sandrojuca@ifce.edu.br>
<leonardo.costa@ppgcc.ifce.edu.br>, <mateus.moura@ppgcc.ifce.edu.br>
<renata@dee.ufc.br>

DOI: 10.21439/jme.v1i1.8

Received: 31 Jan. 2018. **Accepted:** 8 Mar. 2018

Abstract. In this age of automation, the acquisition and monitoring of sensor data is essential to ensure the correct condition and functioning of machines and systems without the need of manual verification. This paper describes the development, assembling and configuration of a system for sensor data acquisition and Cloud-based monitoring. The acquisition system is versatile, allowing the usage from one to six sensors and the replacement of those with little additions to the code. The monitoring system has versions for PHP and Java both on Portuguese and English language. The collected data is read and processed by a microcontroller board and sent to the Internet by a Raspberry Pi. Then, the monitoring system displays it as charts with several different search modes and options to facilitate and give more accuracy for sensor monitoring. All the software is free and open-source, allowing adaptations to suit the user's needs.

Keywords: Internet of Things, sensors, data acquisition, online monitoring.

1 Introduction

The Internet of Things (IoT) is a theme that becomes more popular around the world with each passing day. It's a concept that not only has the potential to impact how we live but also how we work Morgan (2014). The IoT is a deep subject, that in a simplified manner means to connect devices (or even parts of devices) to the Internet and allow them to communicate with each other.

In recent years a growth in the development of data acquisition and digital signal processing is visible (SMAR, 2017), and it is estimated that by 2020 we will have around fifty billion devices connected to the Internet (CISCO, 2011). The monitoring of sensors is essential in many areas, since it enables the automation of delicate processes that require constant monitoring. However, the current monitoring systems usually deal with a small amount of sensors. This project seeks to overcome this issue.

This paper has the objective of demonstrating the development of a versatile system for data acquisition, Cloud persistence and remote monitoring. For that, a microcontroller SanUSB board (with the microcontroller PIC18F2550) and a Raspberry Pi will be used. The data will be stored in the ThingSpeak platform. The monitoring will be performed through a PHP system, responsible for displaying the data available as charts. Since it is a universal system for AD sensors, the project may easily be extended for other related lines of research.

The next sections will approach the related works, the resources used in the development of this system and its functioning. Afterwards the results, final words, suggestions and improvements for future works will be discussed.

2 Related works

Other works with similar objectives have been found, involving data acquisition and monitoring of sensors for both specific and generic purposes.

Andreola et al. (2016) is about sensor data acquisition for the viability study of wind turbines installation in a specific location. A microcontroller of the PIC family is used, and it stores the data collected in an SD card during a year.

Galante & Garcia (2014) is about a system for a replaceable sensor's data acquisition and viewing. It is based on Arduino, and also stores the data in an SD card.

Khera et al. (2016) built a system for real-time data acquisition, monitoring of the data and repositioning of solar panels based in the sunlight received. The system is based on Arduino as well, and stores the gathered data both in a local storage unit and in the ThingSpeak platform, with the use of a GSM modem.

Mutha, Kumar & Pareek (2016) proposed a system for the acquisition of data from several sensors related to the environment (temperature, humidity, and others), based on Arduino. The data collected has been stored in the Cloud, in the IoT server Carriots.

The works above present a limitation in the amount of sensors, with the maximum being four in Mutha, Kumar & Pareek (2016). Another limitation is present in the storage of the data acquired, since data in the Cloud is more accessible than data stored locally, and the Carriots platform is not entirely free.

This document proposes a more versatile prototype. It is able to accept more sensors and uses alternative storage options when offline, thus avoiding the loss of readings. It is also versatile, allowing its sensors to be replaced with only small changes in the code. The monitoring system allows the user to consult the data stored in a more graphic manner, and provides the user with several different search modes and functions not usually available in storage platforms.

Still, it is worth noting that the fact that the related works use fewer sensors is *not* a disadvantage. They use the amount of sensors necessary for their specific purposes. This project does not focus on the amount of sensors alone — it actually focuses on versatility. It seeks to provide a simple, free, open-source, versatile and reusable system that can be easily adapted to suit the user's needs. This system accepts six (or more) sensors and provides the means for customization and even the adding of more functions, while allowing the user to monitor the data of those sensors in a graphic and accurate manner.

3 Resources

This section approaches the resources used in the system's development:

- The microcontroller SanUSB board.
- The RaspberryPi (RPi).
- The sensors.
- The SanUSB tool.
- The WiringPi library for RPi usage.
- The curl tool for sending data.
- The FusionCharts library for chart generation in PHP/JavaScript.
- The ThingSpeak platform, used for data storage.

3.1 SanUSB board, Raspberry Pi and used sensors

The Raspberry Pi is a computer with reduced size and cost produced by the Raspberry Pi Foundation. Despite its reduced size, the RPi possesses the resources and capabilities of a standard computer. In this project, the RPi's function shall be receiving the data from the microcontroller board and sending this data to the ThingSpeak platform for storage. This will be done by a software written in C language, using the libraries WiringPi and curl. As any other general-purpose computer the RPi has an operational system installed, in this case the Raspbian[1], a free operational system based in Debian and optimized for the RPi's hardware.

Although the RPi is a great learning tool as useful as a standard computer for most tasks and widely used in IoT projects, it doesn't have an analog/digital converter. Such a converter — that is not present in most regular computers — is necessary to read the data of sensors that work with analog values. For that reason, the AD converter of the microcontroller PIC18F2550 will be used when reading the sensors. Afterwards, the microcontroller will send the converted data to the RPI through serial communication.

Another advantage of using the RPi is the possibility of adding extensions to the project. Such extensions could enable the addition of actuators to alter the environment's state or trigger alarms, for example, without

[1] https://www.raspbian.org/

changing the acquisition system itself. That may be achieved using threads within the code, another feature enabled by WiringPi.

A microcontroller is a complete computational system, in which are internally included a CPU, RAM, flash and EEPROM memories, I/O pins, besides several other peripherals Jucá & Pereira (2017). The microcontroller PIC18F2550 was chosen for this project because of its sinergy with the SanUSB tool (explained in 3.2). Besides, the microcontroller used offers the advantages of the PIC18 family, such as high computational performance for an economic price and additional high endurance, Enhanced Flash program memory (MICROCHIP, 2006).

In this project, two sensors are used: temperature and luminosity. They will be powered by the RPi, and each one will be read by one of the analog ports of the microcontroller board. Up to six sensors (limit explained in 3.4) may be used simultaneously with only small additions to the source code.

3.2 SanUSB tool

The transferring of programs to microcontrollers is usually done with a specific burner hardware Jucá, Carvalho & Brito (2009). A microcontroller SanUSB board was used to ease this transference, thus speeding up the project's development. The board can me merged to the RPI efficiently and works with the PIC microcontroller mentioned above, which is the reason behind this specific choice. The objective of this board is to make the microcontroller's burning process easier, communicating with the USB port of a computer for burning and powering. It also provides several other useful functions for microcontroller usage, and a schematic of the board's circuit is available in Jucá, Carvalho & Brito (2009).

After the assembling and initial setup of the microcontroller board, it is necessary to use the SanUSB tool to burn the source code into the microcontroller. Before burning, the code is compiled from C to hexadecimal (.hex), required for running in the microcontroller. Another useful mechanic of the SanUSB tool is the option to be used both through user interface and through command lines, which enables the microcontroller to be burned directly by the RPi. The efficiency of the combined use of the microcontroller board with the RPi is demonstrated in several articles, such as Queiroz et al. (2016).

3.3 Used libraries

WiringPi[2] is a GPIO (General-purpose input/output) access library written in C for the BCM2835 processor used in the Raspberry Pi. Although it is written in C — which is the language used in the development of the data acquisition software — this library is available in several other languages. In this project, the WiringPi library is used for configuring, reading and writing the GPIO ports.

Despite the fact that the RPi in this project is already connected to the Internet, the acquisition system needs a way to send the collected data to the ThingSpeak platform. To this end, the tool/data transference library curl[3] is used. As a free and open-source software, the curl library is used via command lines or code to transfer data, and it is used in the data transferences of thousands of applications.

For the generation of charts in the PHP version of the monitoring system, the library FusionCharts was used. Based in JavaScript, it also has a plugin for use in PHP and is capable of generating many types of 2D or 3D charts and even maps based on dynamically defined data. It was chosen for this project because it is easy to learn and use, with several chart models that can be chosen by an user of implementing level.

3.4 ThingSpeak and monitoring system

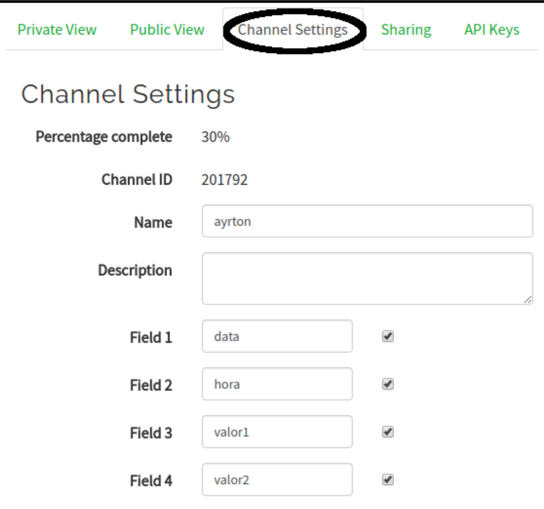
The storage platform selected for use in this project is ThingSpeak[4]. ThingSpeak is an IoT platform that enables the user to store data collected from sensors in the Cloud and develop IoT applications. The platform acts as a system of "channels" that functions like the tables from a database. Each channel has a maximum of eight fields for storage, and two are configured for storing the date and time of readings. The remaining six fields can be used by the sensors. The data sent by the RPi is stored in the ThingSpeak platform so that it may be accessed by the monitoring system by using a shared key.

Configuring a new channel is simple. After creation it is necessary to configure the initial settings (Figure 1) by selecting how many fields should be enabled. Once the required fields are enabled, all that is left is to get the read and write API keys as shown on Figure 2. After the keys are generated, the platform explains how to import and export data to the channel.

[2]http://wiringpi.com/

[3]https://curl.haxx.se/

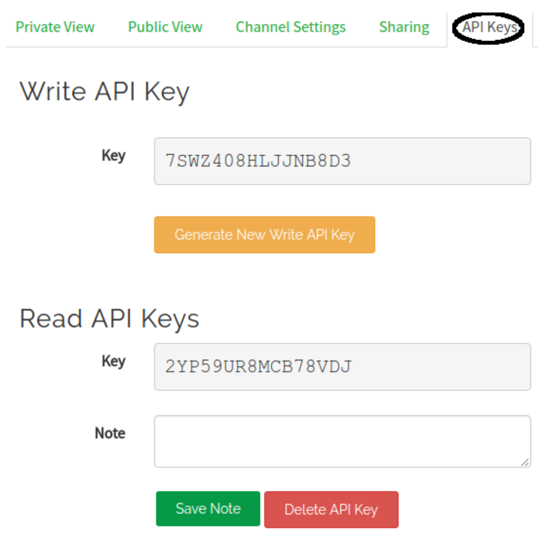
[4]https://www.mathworks.com/help/thingspeak/



Channel Settings

Percentage complete	30%
Channel ID	201792
Name	ayrton
Description	
Field 1	data ✓
Field 2	hora ✓
Field 3	valor1 ✓
Field 4	valor2 ✓

Figure 1: Example of ThingSpeak settings



Write API Key

Key: 7SWZ408HLJJNB8D3

Read API Keys

Key: 2YP59UR8MCB78VDJ

Note:

Figure 2: Example of ThingSpeak API keys

But despite its usefulness, the ThingSpeak platform has certain limitations that make it undesirable for data monitoring: the charts it generates (and the data associated) can only be browsed and by the user to a certain extent. Furthermore, they do not take into account the time the readings were taken, only the time when they were *uploaded*. That means any readings taken on the previous day, for instance, would be accounted by its charts as readings taken today, disregarding the time informed by the user. Still, it does allow the user to retrieve all the data in text form (JSON) through the API mentioned earlier, making it a great storage platform for a user lacking a server of his own.

The monitoring system developed is then responsible for converting the JSON provided by ThingSpeak into the charts viewed by the user. It also provides the user with the means to search through this data (also not provided by ThingSpeak), much like a regular database.

This monitoring system has versions in Java and PHP (with same functions), and is available on Portuguese and English. It needs to be installed in an accessible server so that the charts may always be consulted when needed. The system has a friendly interface, and uses a system of user control based in MySQL.

If the user has programming knowledge he may develop his own monitoring system to replace the one provided, as long as the ThingSpeak API is used correctly to access the data. In the project the account of one of the authors was used for testing, but any user may create his own account. In the moment of channel creation two API keys (read and write) are generated, to be used in requests. Read requests return the data of the channel in JSON format, easy to understand and use. An advantage of choosing ThingSpeak over another platform (such as Carriots) is that all their services are free despite minor limitations. However, the use of ThingSpeak is not mandatory. It may be replaced by another platform, or even by a database in a server managed by the user (removing the dependency on third-party services) with only small changes in the code.

Originally the system was developed only on PHP, but the Java counterpart was developed to prove that anyone with the necessary knowledge can create their own version with its own functions in a short amount of time. The Java version functions exactly as the PHP version (even with some small improvements over the original system) and was developed using JSF and Primefaces. As long as the user uses the ThingSpeak API correctly, the possibilities for customization are many.

4 Functioning

The project is divided into two parts (Figure 3): the embedded Linux system for sensor data acquisition, and a remote system for online monitoring.

The embedded system consists of:

- The microcontroller board (b) left side.
- The Raspberry Pi (b) right side.
- The sensors used (a), connected by a breadboard and a set of wires.

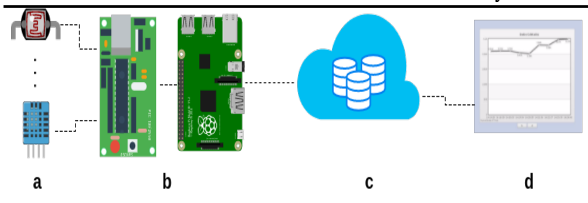


Figure 3: Project schematic

The online monitoring system (d) has a friendly user interface to consult stored data with charts, alongside a security system where the authorized users are stored in a local database.

4.1 Acquisition System

The sensors are powered by the Raspberry Pi, and the read pin of each sensor is connected to one of the analog ports of the microcontroller board. Figure 4 displays the system as it functions: microcontroller board (A), Raspberry Pi (B), sensors — temperature and luminosity — (C) and LEDs (D).

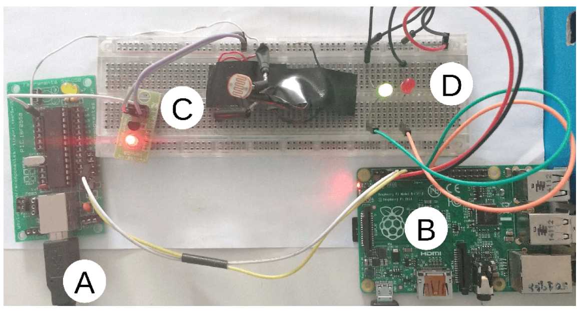


Figure 4: Fully assembled system on breadboard

The PIC reads the voltage values provided by the sensors and converts them to a number between 0 and 1023 (10 bits), where 1023 represents the maximum value that may be read. It is connected to the RPi via USB cable for powering and burning, and via serial (TX and RX) for data transfer.

The PIC takes multiple readings of each sensor connected during one minute, and calculates an average value for each sensor to remove noise and stabilize the readings. After each reading, it uses the serial connection established previously to send the data of each sensor separated by a special character determined on the code.

Upon receiving all readings of that minute, the RPi uses functions from the curl library to send them to a channel on ThingSpeak with a POST request. If the data is sent successfully the green LED is turned on. If there is a failure, the red LED is turned on and the data that wasn't sent is stored locally in the RPi. The data stored cannot be automatically sent once the connection is restored, but it may be inserted on future requests and manually sent to the channel. Figures 5 and 6 display the flowcharts of both stages of the system.

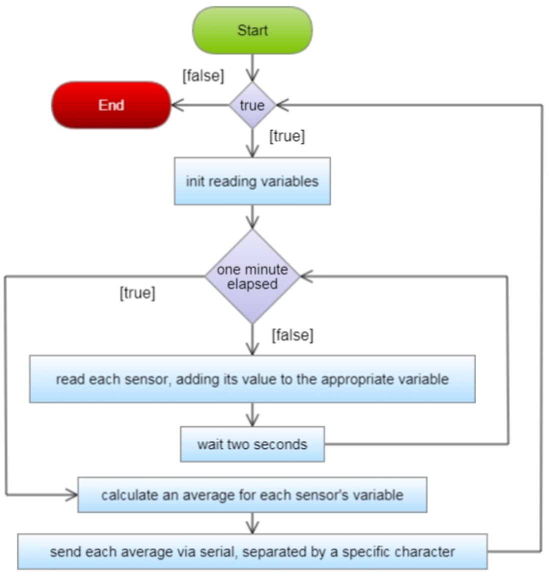

Start
[false]
End
true
[true]
init reading variables
one minute elapsed
[true]
[false]
read each sensor, adding its value to the appropriate variable
wait two seconds
calculate an average for each sensor's variable
send each average via serial, separated by a specific character

Figure 5: Algorithm run on PIC

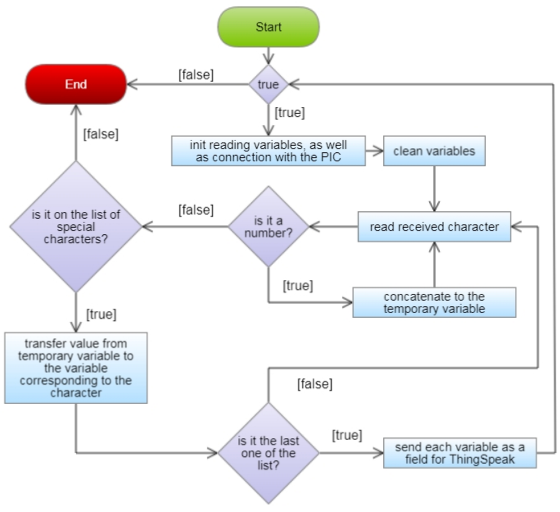

Start
End
[false]
true
[true]
init reading variables, as well as connection with the PIC
clean variables
[false]
is it on the list of special characters?
[false]
is it a number?
read received character
[true]
concatenate to the temporary variable
[true]
transfer value from temporary variable to the variable corresponding to the character
[false]
is it the last one of the list?
[true]
send each variable as a field for ThingSpeak

Figure 6: Algorithm run on RPi

4.2 Monitoring System

The monitoring system developed in this project has the following functions:

- Displaying the acquired data as charts.
- Controlling access to the data.
- Providing the users with functions for searching through the data (not provided by ThingSpeak).
- Allowing the user to monitor the state of the sensors with accuracy.

The data may be consulted by filters where the user can state which dates and times he wants to view. The system also provides a function for data aggregation, that calculates an average of the data during certain periods of time.

Figures 7 through 10 show a basic search in the Java monitoring system. Figure 7 displays the options of the Temperature menu, which contains several different filters for the user. Each filter requires different fields to search through the data.

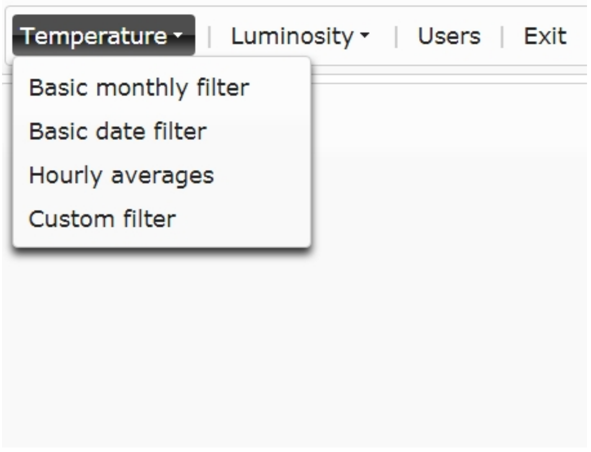
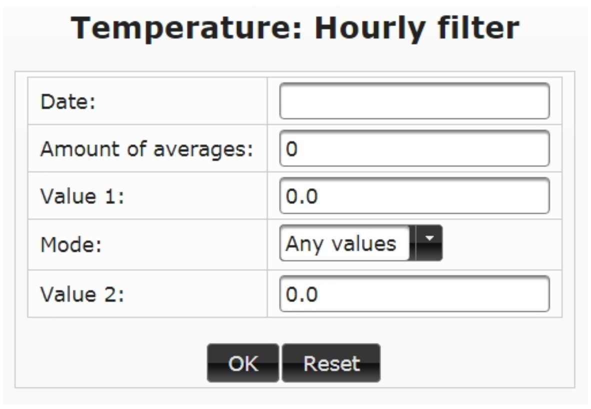


Figure 7: Temperature menu on Java system

Figure 8 displays the screen from the "Hourly averages" menu. It contains a date field to determine the date used and two fields for temperature values. It also contains a mode field, in which the user defines how to use those two values. The available modes are:

- "Any values", which returns the averages from all the readings.
- ">= Value 1", which returns averages from readings where the temperature is greater or equal to Value 1.
- "<= Value 1", which returns averages from readings where the temperature is smaller or equal to Value 1.
- "Between both values" which returns all readings with values ranging from Value 1 to Value 2.

The user also needs do define how many averages will be shown in the chart. On other search modes, the user must also define the field on the X axis of the chart (date, or time) and if the results will be grouped or not. They may be grouped by day or by month, with an average of the selected periods being calculated and displayed.



Temperature: Hourly filter

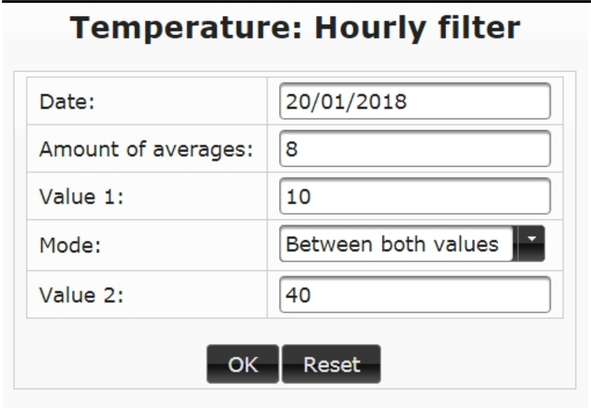
Date:	
Amount of averages:	0
Value 1:	0.0
Mode:	Any values
Value 2:	0.0

Figure 8: Basic date filter on Java system

Figure 9 displays the filter with all fields filled. The "Date" fields use the Brazilian standard, which is DD/MM/YYYY. Eight averages are to be displayed in each chart with the averages being calculated from readings with temperatures between 10 and 40 degrees Celsius.

Figure 10 displays the generated chart. It contains the amount of readings defined, as well as the other configurations. Note that more than the specified amount of readings may be available, in which case the system will provide buttons to navigate through the readings in order to show more information.

Other filters have similar fields which work in the same manner, allowing the user to search based on either date, time or the value of the readings. The system also provides a monthly search in which the user provides the year and the month and the system returns the correct readings.



Temperature: Hourly filter

Date:	20/01/2018
Amount of averages:	8
Value 1:	10
Mode:	Between both values
Value 2:	40

Figure 9: Filled date filter on Java system

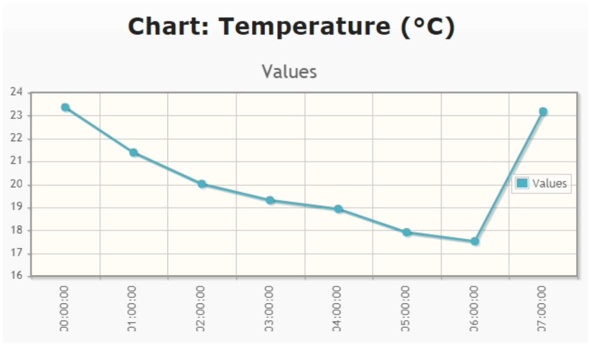


Figure 10: Chart from Java system

4.3 Installation

The acquisition system is fairly straightforward, once assembled. The C codes are available on GitHub, one for the microcontroller board and one for the Raspberry Pi. The first step is to compile the board's code into a .hex file. Then it is necessary to use the SanUSB tool (described in Jucá, Carvalho & Brito (2009)) to burn the appropriate hex file into the board.

Once the board is configured, all that is left is to compile the RPi's code into an executable file and run it.

It is necessary to configure a ThingSpeak channel before using the monitoring system. Both versions of it need to be installed in a properly prepared environment, although the PHP version is easier to install than the Java version.

Some things, however, are needed for both versions. They include:

- MySQL database installed.
- A database named "sis_amd" (name may be changed by the user).
- A username and a password for the system to access the database (defined by the user on the code)
- A table named "usuarios" with the fields:
  - id_usuario (int)
  - nome (varchar)
  - login (varchar)
  - senha (varchar)
- Obs: All these configurations may also be changed by a more experienced user.

Also, for the PHP system the user has to insert the first user into the database in order to log into the system and have the MySQLi extension installed. For the Java system there is a class named CreateDB which will create the tables in an empty database and insert an initial user, that can be defined on the code.

Once the initial configuration is done, one needs simply to place the PHP folder inside the server and access it. The Java version, however, requires being built first. The user may build it in the latest version of the Eclipse[5] IDE, as long as it is configured for Java EE. Then, he can run it from Eclipse itself or build a WAR file to post in a server.

5 Results

While ThingSpeak provides charts that cannot be effectively browsed by the user (Figure 11), the monitoring system developed in this project allows the user to search through *all* of the data stored. It enables the user to perform searches through the data based on dates, times and values (Figure 12), and even calculate averages for certain periods of time (Figures 13 to 16).

Figures 13 and 14 display temperature and irradiance readings (respectively) collected during a sunny day (January 21st), presenting temperatures ranging from 25 to 33 degrees Celsius and irradiance values that reached 992 W/m^2 around noon.

Figures 15 and 16 display readings of the same sensors during a cloudy day (January 23rd). The highest temperature is two degrees lower than the highest temperature from the previous chart, but the difference between the irradiance values is greater. Irradiance values on the 23rd remained low during all day, reaching a maximum of 587 W/m^2 near noon.

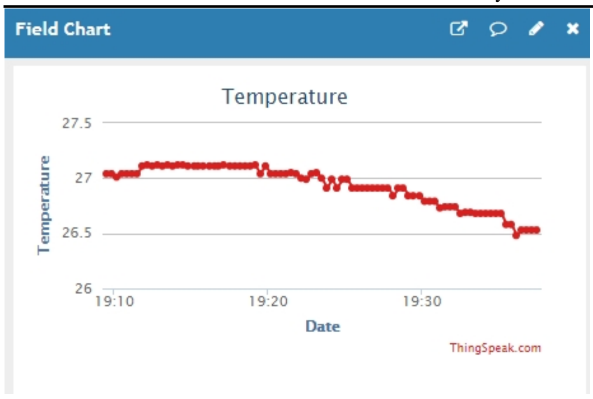
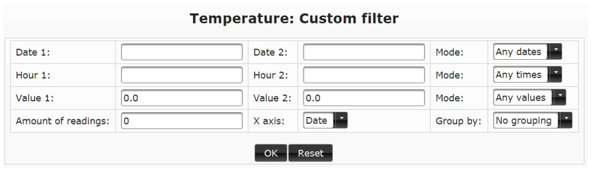


Figure 11: ThingSpeak chart



Temperature: Custom filter

Date 1:		Date 2:		Mode:	Any dates
Hour 1:		Hour 2:		Mode:	Any times
Value 1:	0.0	Value 2:	0.0	Mode:	Any values
Amount of readings:	0	X axis:	Date	Group by:	No grouping

OK Reset

Figure 12: Filter screen

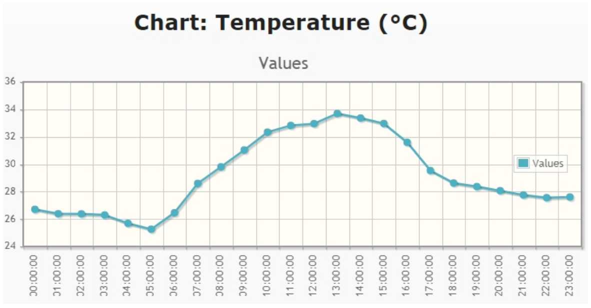


Figure 13: Chart with hourly averages for temperature on a sunny day

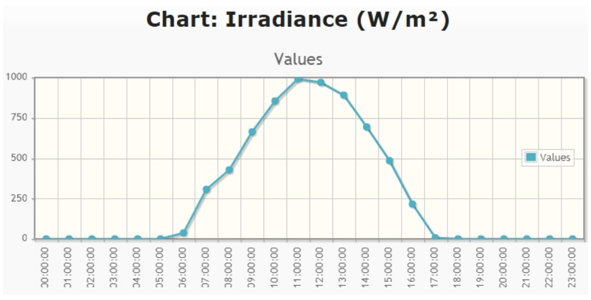


Figure 14: Chart with hourly averages for irradiance on a sunny day

All the data displayed on the previous charts has

[5]https://www.eclipse.org/

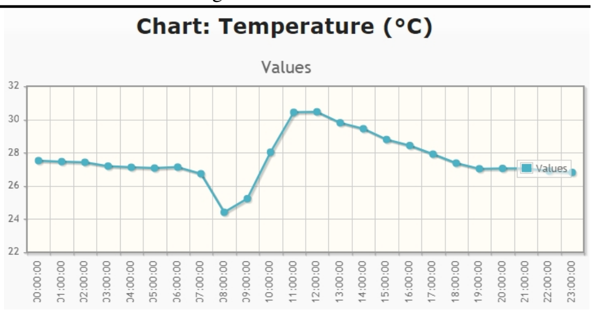


Figure 15: Chart with hourly averages for temperature on a cloudy day

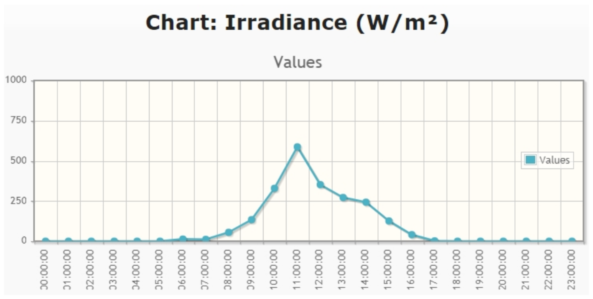


Figure 16: Chart with hourly averages for irradiance on a cloudy day

been collected by the Laboratory of Alternative Energies (LEA) from the Federal University of Ceará (UFC), using temperature and irradiance sensors.

Screen shots of both systems are available in a shared folder on Dropbox Silva & Costa (2017a). A video on youtube Silva & Costa (2017b) features the functioning of the acquisition system, and all the source code developed is available on GitHub Silva & Costa (2017c).

6 Conclusion

This project differs from others in the same line of research because of its purpose. This project intends to provide a free, open-source, simple, versatile and reusable data acquisition and monitoring system, while other projects are directly focused on simply collecting and monitoring the data from their sensors efficiently. This system may be adapted and customized according to the user's needs and is able to access a larger number of sensors.

In that regard, the project was a success. The acquisition system works as it should, accepting a large range of sensors that can be replaced by the user in a short amount of time and displaying the data collected in a concise manner. The monitoring system has a PHP version and a Java version, both available on Portuguese and English. It provides a number of functions unavailable on ThingSpeak which facilitate and give more accuracy to the monitoring process. Thus, ThingSpeak is better used as a storage platform, rather than a monitoring platform. Since the monitoring system acts upon the API provided by ThingSpeak, it may be easily adapted and/or replaced by the user, as proved by the development of alternate versions.

The project has good extension potential for use with a variety of sensors, since many of them can be read by the PIC and their analog data converted to digital values with few additions to the code. The use of Raspberry Pi enables the possibility of even more functions being added to the system and divides responsibilities with the microcontroller, making the code burned into it much simpler.

And it is exactly that extension potential that brings ideas for future projects. Examples would be a system that uses the collected data for making decisions (such as triggering alarms), or a system that automatizes the process of adding/removing sensors from the acquisition system by generating code, making adaptations more dynamic.

References

ANDREOLA, A. T.; SENTER, M. J. D.; TODERO, E. L.; CARDOSO, G. Low cost data acquisition system for wind prospecting. In: **2016 12th IEEE International Conference on Industry Applications (INDUSCON)**. [S.l.: s.n.], 2016. p. 1–6.

CISCO. **The Internet of Things**. 2011. Disponível em: <http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf>. Acesso em: 23 jul. 2017.

GALANTE, A. C.; GARCIA, R. F. Sistema de aquisição de dados de sensores de baixo custo baseado no arduíno. In: **Congresso Brasileiro de Agricultura de Precisão**. [S.l.: s.n.], 2014. v. 6.

JUCÁ, S. C.; CARVALHO, P. C.; BRITO, F. Sanusb: software educacional para o ensino da tecnologia de microcontroladores. **Ciências & Cognição**, v. 14, n. 3, 2009. ISSN 1806-5821. Disponível em: <http://www.cienciasecognicao.org/revista/index.php/cec/article/view/254>.

JUCÁ, S. C. S.; PEREIRA, R. I. S. **Aplicações Práticas de Microcontroladores utilizando Software Livre**. 1. ed. [S.l.]: Imprima, 2017.

KHERA, N.; SINGH, S.; SHARMA, A.; KUMAR, S. Development of photovoltaic module tracking and web based data acquisition system. In: **2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)**. [S.l.: s.n.], 2016. p. 100–103.

MICROCHIP. **PIC18F2455/2550/4455/4550 Data Sheet**. 2006. Disponível em: <http://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf>. Acesso em: 23 jul. 2017.

MORGAN, J. **A Simple Explanation Of 'The Internet Of Things'**. 2014. Disponível em: <https://goo.gl/Yg9cEY>. Acesso em: 23 jul. 2017.

MUTHA, V. R.; KUMAR, N.; PAREEK, P. Real time standalone data acquisition system for environmental data. In: **2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)**. [S.l.: s.n.], 2016. p. 1–4.

QUEIROZ, T. A. F.; DIAS, D. L.; ARAÚJO, P. H. M.; FIGUEIREDO, R. P.; JUCÁ, S. C. S. Sistema embarcado linux para análise de sensores de temperatura dht11 e lm35. **ERIPI 2016**, 2016.

SILVA, A. A. M.; COSTA, L. S. 2017. Disponível em: <https://www.dropbox.com/sh/fnsnvhqgq1xhyrt/AAAXItAk386cTAV8C628sga0a?dl=0>. Acesso em: 28 jul. 2017.

____. 2017. Disponível em: <https://www.youtube.com/watch?v=j68Pd2bpm3M>. Acesso em: 28 jul. 2017.

____. 2017. Disponível em: <https://github.com/Ayrtonms>. Acesso em: 28 jul. 2017.

SMAR. **Sistemas de Supervisão e Aquisição de Dados**. 2017. Disponível em: <http://www.smar.com/brasil/artigo-tecnico/sistemas-de-supervisao-e-aquisicao-de-dados>. Acesso em: 23 jul. 2017.