

Design and Implementation of Modular Environmental Observing Stations Using Off-the-Shelf Components

Artur Nogueira
MI:ERSI
Departamento de Ciência de Computadores
2018

Orientador
Luís Lopes, Professor associado, FCUP

Co-orientador
Sérgio Crisóstomo, Professor auxiliar, FCUP

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

Abstract

Nowadays, there is accessible and affordable hardware to assemble sophisticated embedded systems. This is something which could not be said several years ago. In the modern days, we are able to build very tiny systems with tremendous amount of computing power and very low power consumption. In comparison, in the past, embedded systems were very expensive to build, occupied a lot of physical space and were created for very restricted purposes. Technology evolution has provided us with tools to design and implement more complex and flexible real-life projects. However, even now, the great majority of embedded systems that exist are somewhat limited for what they were created for and re-purposing them involves a complex process of altering/adapting both hardware and software in order for the new features to work. Specifically regarding our project, weather stations are a good example. Usually, commercial weather stations integrate hardware perfectly capable of extending the standard meteorological equipment, but the bundled software is very narrow and does not support personalization.

Our intention was to create modular Environmental Observing Stations (EOS). These stations would be deployed on a certain location with the goal of harvesting environmental data. A station could be assembled for a wide range of case scenarios (e.g., monitoring weather, air quality or seismic activity, observational astronomy, surveillance of natural habitats, etc). These stations should offer flexibility to be easily customizable and suited for different environmental schemes, i.e., readjusting an EOS would not require many complex operations and/or technicalities. Thus, the developed software system should have a modular architecture in order to support this level of modifications. Furthermore, non-specialist users should be able to configure and operate an EOS with ease. These EOS would be built having a low-cost perspective in mind, assembled using off-the-shelf components.

We designed our system to support multiple EOS. We have a centralized server, which is responsible for communicating with each deployed station and performing several logic tasks. We have created an Environmental Observing Station prototype with the purpose of doing astronomical and meteorological observation. All the hardware used is based on off-the-shelf components. Its physical structure was assembled from scratch, providing a unique aesthetics to the station, while being a low-cost solution.

After we assembled our EOS, we submitted it to a period of experimentation. We set it up on a yard for a month. The purpose was to test the viability of the physical structure and check how

well it performed under direct heat of sunlight and rainy conditions. Additionally, it was a way to fully test the capturing software with the acquired devices, as well as, the data synchronization process with the server. After the troubleshooting stage, we deployed our prototype station at Parque Biológico de Gaia [1]. There were several requirements, which needed to be accomplished in order for the EOS to be operational - fixation of the main structure in place, power and network cable routing and network access configuration. On the 17th of July, our station went online and has been functioning ever since. It is capturing temperature, humidity, pressure and luminosity data and performing astronomical observation during the night. It is also uploading data to the Weather Underground service [2].

[1] http://www.parquebiologico.pt
[2] https://www.wunderground.com/weatherstation/overview.asp

Acknowledgements

I would first like to thank my thesis advisors Prof. Luís Lopes and Prof. Sérgio Crisóstomo. Their guidance and insightful suggestions throughout the year were very helpful. Plus, our weekly meetings allowed me to solve many encountered problems along the way, while keeping on the right track in the development of this project.

I would also like to express my gratitude to my father. His aid in the assembly of the physical structure was fundamental. We spent many hours in the garage building it, but I think the end result was worth it. I do not think I would be able to accomplish it on my own. Thank you.

I would like to thank INESC TEC and the Instituto de Telecomunicações for their support in carrying out this work.

Dedication

I would like to dedicate this work to my parents, especially my mother. She has always provided me encouragement and support throughout my academic journey. I hope I have made her proud.

Contents

1 Introduction **1**

1.1 Problem Statement 1

1.2 Document Structure 3

2 Technical Background **5**

2.1 Communication Protocols 5

2.1.1 USB 6

2.1.2 I^2C 6

2.1.3 One-Wire 7

2.1.4 SPI 8

2.2 Embedded Software Development 8

2.3 Web Development 9

2.3.1 Model–View–Controller (MVC) 10

3 State-of-the-Art **13**

3.1 Computing Platform 13

3.1.1 Arduino 14

3.1.2 Raspberry Pi 15

3.1.3 BeagleBoard 15

3.2 Sensors 17

3.2.1 Astronomy Cameras 17

3.2.2 Temperature and Humidity 18

3.2.3 Wind Speed and Direction . . . 19
3.2.4 Rain Intensity . . . 20
3.2.5 Atmospheric Pressure . . . 20
3.2.6 Luminosity . . . 21
3.2.7 Air Quality . . . 22
3.2.8 Position Tracking . . . 23
3.3 Power Availability . . . 23
3.4 Connectivity and Communication . . . 26
3.5 Available Solutions . . . 29
3.5.1 Astronomy . . . 29
3.5.2 Meteorology . . . 31

4 Architecture **35**
4.1 Requirements . . . 35
4.2 Overview . . . 36
4.3 Server . . . 39
4.4 Station . . . 41
4.5 Concrete Example . . . 43

5 Implementation **47**
5.1 Server . . . 48
5.1.1 Database . . . 48
5.1.2 Content Visualization . . . 50
5.1.3 Content Management . . . 53
5.2 Station . . . 58
5.2.1 Physical Structure . . . 60
5.2.2 Station Preparation . . . 63
5.2.3 Station Configuration . . . 64
5.2.4 Capturing Data . . . 69

5.2.5 Weather Underground Integration 71

6 Discussion **75**

6.1 Astronomy Camera 75

6.2 Temperature 82

6.3 Humidity 85

6.4 Atmospheric Pressure 86

6.5 Luminosity 87

6.6 Weather Underground 88

7 Conclusions and Future Work **91**

References **93**

List of Tables

3.1 Table of Processing Platforms. 16

3.2 Table of specs for ZWO ASI120MM. 18

3.3 Table of Humidity/Temperature sensors. 19

3.4 Table of digital barometric air pressure sensors. 21

3.5 Table of luminosity sensors. 22

3.6 Table of LPWAN technologies. 28

5.1 Table of “Kit Insert” list of API’s request. 56

5.2 Table of “Kit Configuration” RESTful API. 67

List of Figures

2.1 I^2C interface (from [35]). 6

2.2 One-Wire interface (from [50]). 7

2.3 SPI interface (from [71]). 8

2.4 MVC Design Pattern (from [47]). 10

3.1 Arduino Uno Rev3 (from [7]). 14

3.2 Raspberry Pi 3 Model B (from [59]). 15

3.3 BeagleBoard Black (from [11]). 15

3.4 ZWO ASI120MM (from [8]). 18

3.5 Humidity/Temperature sensors. 19

3.6 Anemometers (left to right: ultrasonic, cup and wind vane) (from [4]). 20

3.7 Raspberry Pi Rain Gauge Sensor (from [26]). 20

3.8 Digital barometric air pressure sensors. 21

3.9 Luminosity sensors. 22

3.10 TGS 2600 Gas sensor (from [74]). 22

3.11 MTK3339 GPS module (from [30]). 23

3.12 Example of li-polymer and solar panel powerbanks. 24

3.13 Example of a USB voltage step-down regulator (from [19]). 24

3.14 Solar powering system for DC and AC devices (from [70]). 24

3.15 UPS PIco - Uninterruptible Power Supply (HV3.0B+) (from [77]). 25

3.16 GSM relay switch (from [32]). 25

3.17 Sigfox network architecture (from [67]). 27
3.18 Sigfox wireless shields (from [68]). 27
3.19 LoRa wireless shields (from [40]). 27
3.20 Digi XBee Cellular NB-IOT (from [23]). 28
3.21 Images from project [37]. 29
3.22 Oculus all-sky camera [48]. 30
3.23 OMEA system [49]. 30
3.24 Davis Instruments Solutions. 31
3.25 Raspberry Pi Air Sensor Board and Weather Board (from [60]). 32
3.26 AirPi (from [2]). 32
3.27 SwitchDoc Weather Board (from [82]). 33

4.1 Diagram of components of the global system. 36
4.2 Diagram of software components of the server. 37
4.3 Diagram of software components of an EOS. 37
4.4 Diagram of hardware components of an EOS. 38
4.5 “Kit Insert” Conceptual Diagram. 39
4.6 “Kit View” Conceptual Diagram. 39
4.7 Diagram of communication between users and the server. 40
4.8 “Kit Insert” Data Flow Diagram. 40
4.9 “Kit View” Data Flow Diagram. 41
4.10 “Kit Config” Conceptual Diagram. 42
4.11 “Kit Config” Data Flow Diagram. 42
4.12 Capturing Software State Machine Diagram. 43

5.1 Communication diagram between users, the EOS and the server. 47
5.2 Database Relational Diagram. 49
5.3 “Kit View” PBG - Temperature chart. 51
5.4 “Kit View” PBG - Camera’s latest picture. 52

5.5 "Kit View" - All deployed EOSs overview . . . 52
5.6 "Kit Insert" - Table of Default Configurations. . . . 57
5.7 "Kit Insert" - Edition form of a configuration. . . . 57
5.8 Station Deployment. . . . 59
5.9 Astronomical Observatory at Parque Biológico de Gaia. . . . 59
5.10 Solar radiation shield assembly. . . . 60
5.11 Connection of sensors inside the solar radiation shield. . . . 61
5.12 Homemade solar radiation shield. . . . 61
5.13 ZWO ASI120MM support platform assembly. . . . 62
5.14 Camera support platform. . . . 62
5.15 Control panel. . . . 63
5.16 Local Database Relational Diagram. . . . 64
5.17 "Kit Configuration" - Configuration Interface (TSL2561 device). . . . 65
5.18 "Kit Configuration" - Home Page (Sync Request and WUnderground access). . . . 65
5.19 "Kit Configuration" Android app. . . . 69

6.1 All-sky picture (29 of April at 07:52). . . . 76
6.2 All-sky picture (29 of April at 09:03). . . . 76
6.3 All-sky picture (27 of July at 18:23). . . . 77
6.4 All-sky picture (28 of July at 12:12). . . . 78
6.5 All-sky picture (25 of July at 01:07). . . . 79
6.6 All-sky picture (31 of July at 03:58). . . . 79
6.7 All-sky picture (5 of August at 04:20). . . . 80
6.8 All-sky picture (8 of August at 01:14). . . . 81
6.9 All-sky picture (20 of August at 04:23). . . . 81
6.10 Temperature Chart. . . . 83
6.11 Temperature Difference Chart. . . . 84
6.12 Scatter Plot of Temperature. . . . 84

6.13 Humidity Chart. 85

6.14 Atmospheric Pressure Chart. 87

6.15 Luminosity Chart. 88

6.16 PBG - Weather Underground. 89

Acronyms

ADC Analog-to-Digital Converter. 17, 21

API Application Programming Interface. xi, 5, 10, 26, 28, 36, 38, 41, 48, 54–56, 64, 67, 69, 70

CCD Charge Coupled Device. 17

CMOS Complementary Metal-Oxide Semiconductor. 17

CMS Content Management System. 54

CSRF Cross-Site Request Forgery. 53

DBMS Database Management System. 54

EOS Environmental Observing Station. i, ii, xiv, xv, 1–3, 13, 14, 16, 23, 26, 35–41, 43–45, 47, 48, 50, 52–54, 56, 58, 63, 64, 69, 75, 77, 89–92

GPIO General-Purpose Input/Output. 8, 17

HTTP Hypertext Transfer Protocol. 41, 52, 53, 71

I^2C Inter-Integrated Circuit. xiii, 2, 5–8, 17, 19, 21, 36, 44, 51, 58, 60, 63, 65, 66, 69, 91

IDE Integrated Development Environment. 9, 14

IoT Internet of Things. 28

LPWAN Low Power Wide Area Network. 26–28, 35, 44, 92

MVC Model–View–Controller. xiii, 10, 11, 38, 48

RTOS Real-Time Operating System. 9

SBC Single Board Computer. 13, 15, 63

SDK Software Development Kit. 18, 29, 63, 66, 70

SoC System on Chip. 13

SPI Serial Peripheral Protocol. xiii, 5, 8, 19, 21, 58

UPS Uninterruptible Power Supply. 25

USB Universal Serial Bus. 5, 6, 15, 18, 23, 24, 36, 44

Chapter 1

Introduction

Nowadays, sensor technology is used in several areas (e.g., medical, energy sources, environmental monitoring) with many practical developments. Harvesting data from the physical world allows to study it and better understand it. In the past, the ability to sense natural phenomena required a network of sensors with specific hardware and software. Today, there is a wide range of sensors integrated in smartphones, home appliances and vehicles. There are also available platforms, such as Arduino [5] and Raspberry Pi [58], which offer the possibility to easily connect sensors through analog and digital interfaces and create embedded systems for a variety of field applications. However, many projects build systems for specific case scenarios and, sometimes, require too much reconfiguration in order to function for other purposes. Commercial weather stations, for example, consist in a pre-defined set of sensor devices. Normally, users are unable to easily add/remove new components either because the included software does not support customization or the physical system is only prepared for the included devices. Thus, it is important to design flexible systems capable of being easily customized and assembled for different scopes by non-specialist users.

1.1 Problem Statement

We propose a software framework for assembling Environmental Observing Stations (EOS) based on off-the-shelf components. The main goal is to build a modular and low-cost system so it can be applied in different environmental fields. As case-studies, there are two main categories we intend to build a station for: astronomy and meteorology. The station must support (by software) a customization of its connected components, i.e., provide mechanisms for controlling devices (activating/deactivating) and configuring their usage settings, individually. Furthermore, the software should be compatible with a re-adaptation of the EOS. This re-adaptation can be the replacement of certain devices or the addition of new ones. The process involved in a re-adaptation should not be too complex in order to be performed by non-expert users. Though, the user must have some hardware and programming knowledge. The necessary steps performed by a user (e.g., hardware connection, device drivers installation) must be followed by an auto-

detection of plugged components from the software. These requirements are essential for having a flexible system which a non-specialist user should be able to operate with ease.

We aimed to build a modular system in order to support a unique customization of each deployed station. According to a desired environmental scenario and specific requirements (associated with the deployment location and the type of data to be collected), an EOS would be assembled. Depending on the deployment location, there may be constraints in energy autonomy and communications, which should limit how the station is assembled and which components are used. The embedded system would contain a combination of selected components supported by the software framework and based on the harvesting goals. Also, a proper physical structure would be built in order to operate on variable atmospheric conditions, allowing a correct positioning of the sensors on the outside and a good insulation for the preservation of the electronic components inside.

As the implementation of our proof-of-concept, we present a system capable of supporting multiple deployed EOS, with each operating autonomously. There is a centralized server, responsible for communicating with online stations and providing certain services. We have created a prototype EOS, which is currently deployed at Parque Biológico de Gaia. Its physical structure is homemade and all the integrated components are based on off-the-shelf hardware. The station contains a few sensor devices for measuring weather properties (temperature, humidity and pressure), luminosity and monitoring the sky (using an astronomy camera).

Our prototype integrates a Raspberry Pi as the main computing platform, where all the sensor components are connected. These devices communicate through USB, I^2C or One-Wire interfaces. The usage of addressable communication protocols (such as I^2C and One-Wire) allows our software to identify each plugged component and map its ID (address) to relevant stored information. The station runs several software components on top of a Linux operating system. The software system in each station is mainly responsible for the following tasks:

- Perform an auto-detection of components: identification and mapping of a component; from the address we get information about the device (its specifications, configuration settings, etc);
- Provide a configuration interface system: an application from which a user can see the devices recognized by the system and set their usage configurations (e.g., set capture cadences);
- Execute the capturing software: based on the recognized devices and their configuration parameters, perform data harvesting;
- Request a database synchronization: the local database stored in each EOS is derived from the central database (information about devices necessary for the detection process and default configurations). An online EOS can request a new database synchronization (e.g., a new device was added to the server and is intended to be used by the station).

Besides the EOS, our system contains a centralized server. It is the architectural component which supports a central view of multiple deployed stations. The server is responsible for interacting with each deployed station (the ones having Internet connectivity) and providing the following services:

- Centralized content management: a central database containing information about supported devices, deployed EOS and validated data collected by the stations. An application can be used by the user to view and manage some of the content;
- Data synchronization with each EOS: the server retrieves the data dumps generated by each EOS (e.g., log files, images), processes the collected data and stores it in the central database;
- Provide visual content per EOS: an application offers a display of visual content per station (e.g., a time series chart for each harvesting device).

Our developed work allows us to continue to investigate new case studies. Due to the flexibility of our system, we can create additional Environmental Observing Stations in the future and explore different environmental schemes. All the data being collected may be used in external projects for further analysis. For instance, the astronomy images retrieved can be processed to detect celestial events (e.g., meteor shower counts, optical astronomical transients, light pollution gauge) and the weather data can serve as input in weather forecast systems (climate prediction algorithms).

1.2 Document Structure

The remainder of the report is organized as follows: in Chapter 2, we introduce some technical concepts necessary for the development of this project. In Chapter 3, we overview the state-of-the-art in terms of hardware and software frameworks available for development of EOS. Based on this information, we describe and justify the choices of the hardware and software platforms we used. Chapter 4 contains a description of the architecture, giving a conceptual overview of the implemented system. In Chapter 5, we describe the implementation of a proof-of-concept, giving a thorough description of the technical aspects of the implementation. Chapter 6 reports the experimental results and analysis during the time period the station was deployed. Finally, Chapter 7, presents some preliminary conclusions and discusses difficulties encountered while implementing the proof-of-concept, as well as, current issues to be dealt within the future. We also address possible future work related to the creation of more EOS for more environmental scenarios.

Chapter 2

Technical Background

In this chapter we introduce some background material, explaining communication protocols and technical concepts used in the development of this project.

There are several protocols used for the communication between circuit boards and external peripherals. Since our stations integrate a Raspberry Pi Model 3 as its computing platform, we use USB and I^2C/One-Wire devices. The Raspberry Pi also has a Serial Peripheral Protocol (SPI) bus, but it is limited to two SPI devices connected simultaneously. I^2C and One-Wire protocols assign unique addresses for each device, hence it is favorable for our platform to recognize and map the connected components. In case of using a device which does not support I^2C/One-Wire, whether because it is analog or uses some other digital protocol, we can use an adapter/converter to those protocols and assign specific addresses. These adapters are usually low-cost and can be easily acquired together with the sensor devices.

In software development, we communicate with the hardware peripherals through their software drivers. The driver's job is to allow software to communicate with the device's hardware. The software communicates with the device driver through an API and the driver sends commands to the device's controller. Usually, the device's manufacturers provide the appropriate drivers and APIs to easily interact with the devices. However, sometimes it is necessary to edit that low-level software and adapt it for a particular usage, as was our case with one of the sensors used. Moreover, in the worst case scenario, it is possible that some devices do not have drivers available. Thus, the software driver's implementation requires a thorough analysis of the device's datasheet to know how the device operates internally.

2.1 Communication Protocols

Communication protocols define how data is exchanged between individual components electrically connected. There are several protocols grouped into two categories: serial and parallel. The difference between the two is that serial protocols can only transfer a bit of data at a time whilst parallel can transfer multiple bits simultaneously. Typically, parallel protocols achieve

higher data rates than serial, but use more communication channels and require more complex processing. Embedded system boards generally have support to a few serial/parallel protocols. We will now give an overall description of the communication protocols most commonly used.

2.1.1 USB

Universal Serial Bus (USB) is a serial communication protocol developed in the 1990s. It is the most used standard already with several versions and different connectors available [79]. A wide variety of available peripherals utilize USB, from storage drives and smartphones to digital cameras. Devices communicate in a star topology, where a device (usually a computer) is the central host containing one or multiple USB controllers with several ports where all the other devices are plugged. The data transfer is done through a channel (also known as pipe) and there are 4 different transfer modes: control, isochronous, interrupt and bulk modes [80]. A USB connection has 4 wires: the +5V and ground for power and two data signals (known as D+ and D-) [78]. USB can be used as a powering mechanism to charge devices or for information exchange. Nowadays, the data rates vary from 480 Mbps (revision 2.0) up to 10 Gbps (revision 3.1 Gen 2). For example, the astronomy camera used in this project has an USB 2.0 connection (ZWO ASI120MM [8]).

2.1.2 I²C

Inter-Integrated Circuit (I²C) is a bus communication protocol developed by Philips (now NXP Semiconductors) [64]. The communication is done between the master (the controller on the mainboard) and the slaves (each device connected). I²C supports interaction between multiple masters to multiple slaves (Figure 2.1) with collision detection and auto re-transmission. I²C interface has 4 wires: the powering pair (+5V and ground) and a pair of signal wires (data and clock named SDA and SDK, respectively). I²C is a half-duplex synchronous protocol where the master uses the clock signal to give turns to the devices to use the bus.

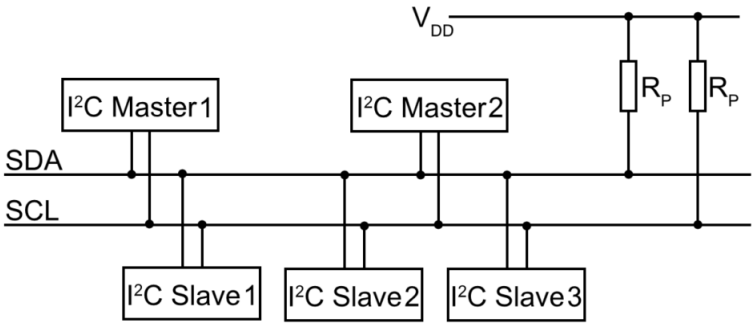


Figure 2.1: I²C interface (from [35]).

Each device is assigned a unique address and they all share the same wires. Initial versions of I²C allowed addresses of 7 bits, that is, up to 128 simultaneously connected devices using unique

ID's. Each I^2C device model has a fixed default address which is assigned by the manufacturer after NXP consent [1]. That way, it facilitates the usage of devices without the need to change configurations, avoiding collisions of the same address on different device models. Furthermore, some devices have the possibility of choosing between several addresses to use, setting them through a jumper on the specified pins. With the growth of I^2C devices, subsequent revisions of the protocol pushed the address support to 10 bits, accepting 1024 unique IDs. In terms of speed, the latest revisions of the protocol support data rates of 100 kbps, 400 kbps and 3.4 Mbps [36]. Many available and affordable digital sensors can use the I^2C interface. In this work, the BME280 [12] and TSL2561 [75] devices use I^2C.

2.1.3 One-Wire

One-Wire is a serial communication protocol which was created by Dallas Semiconductor (now Maxim Integrated). One-Wire bus has a master/slave architecture allowing only a single master. The communication between the master and the slaves is half-duplex and asynchronous. One-Wire devices uses only 1 data signal [51]. Analogously to I^2C, One-Wire devices have assigned a unique ID, of 64 bits, and they all share the same channel. One-Wire devices have two ways of being powered: use the typical separate voltage supply wire, or share the same wire for both power and data transfer. The latter is known as parasitic mode, in which the device integrates a capacitor which stores energy that is used when the wire is transferring data. That way, the data communication and powering modes are switched periodically, while the sensor is maintained on. One-Wire addresses are factory-programmed on the device's ROM and are unalterable. The protocol uses the first 8 bit subset of the addresses to group devices into families of types (e.g., temperature, pressure).

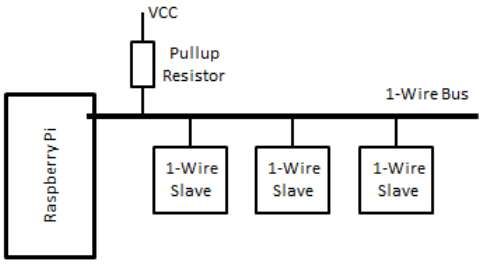


Figure 2.2: One-Wire interface (from [50]).

The physical connection of One-Wire devices requires the use of a pull-up resistor of 4.7 Ohms between the data signal and the voltage supply line, as illustrated in Figure 2.2. In this project, we use an One-Wire temperature sensor (model DS18B20 [27]).

2.1.4 SPI

The Serial Peripheral Protocol (SPI) was created by Motorola. It is an interface bus for synchronous communication between embedded platform's micro-controllers and external peripherals. SPI has a master/slave configuration, having one master and a certain number of slaves. SPI uses 4 signal wires: a clock line (SCK), a data line for each direction: from master to slave (MOSI, Master-Out Slave-In) and from slave to master (MISO, Master-In Slave-Out) and the slave select line (SS), used to activate the communication with the selected device. SPI is a full-duplex protocol, meaning the devices can send and receive data at the same time, by using separate lines for sending and receiving [72]. The number of slaves is limited to the number of SS lines on the computing platform, as one is necessary per device (Figure 2.3). Although, some platforms allow assigning other GPIO pins as SS lines (as is the case of the Raspberry Pi).

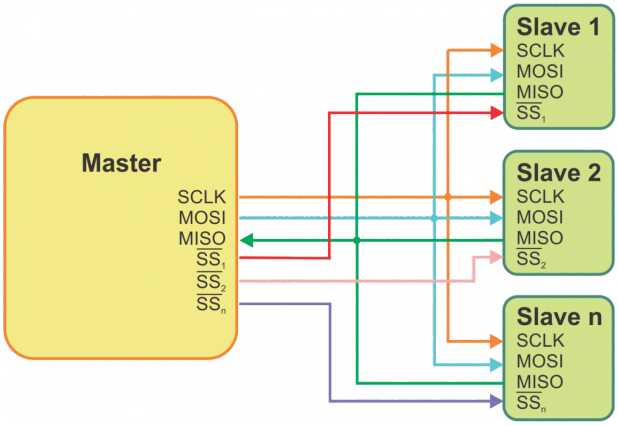


Figure 2.3: SPI interface (from [71]).

Generally, the devices which support the SPI interface also support I^2C.

2.2 Embedded Software Development

Embedded systems are computing devices designed to perform specific and dedicated tasks. They consist of a circuit board integrating a micro-controller (or micro-processor), volatile memory (RAM), flash storage (for the firmware and executable routines) and interfaces (for connecting external peripherals). Depending on the tasks to be executed, the embedded system is designed with the necessary hardware resources to maximize performance and minimize power consumption. Embedded systems can be found in an ample range of electronics devices, such as smartphones, cars and home appliances. These electronic boards are responsible for controlling parts of the machine's execution and, on some equipment, allow users to configure and change how the device operates (e.g., set different speed modes). Nowadays, there are several embedded computing platforms available for building convenient real-life projects. One of the most successful areas where embedded systems are applied in is the world of Internet of Things.

An embedded system is not a general purpose computer. Typically, it does not run a general purpose operating system like Windows and Linux. Instead, it runs its own firmware/operating system. The OS combines scheduling algorithms which are prepared for certain fixed and event-triggering tasks (those events can be automated or from user interaction). In other words, the scheduler has a predictable behaviour. In practice, the execution works as an infinite loop, running scheduled or event-driven tasks with a deterministic execution pattern. Frequently, embedded systems use a Real-Time Operating System (RTOS) [62], designed to meet real-time requirements. That is why it is import to do requirements gathering and establish constraints prior to the system's design. Programming software for an embedded system requires some level of knowledge of the hardware to be used and, sometimes, some low-level software development, depending on the target system. Nowadays, platforms like Arduino, ease embedded software programming with frameworks that offer some abstraction from some low-level concerns and allow the developers to only focus on the application features. Hardware platforms like the Raspberry Pi are even capable of running operating systems with user interaction. Thus, embedded software development is not restricted to a specific programming language or programming paradigm, allowing the creation of more complex embedded projects.

The development of software for embedded platforms is done on a host computer and then flashed into the target system where it is executed. The source code is passed through a cross-compiler in order to generate the binary code to be run on the platform's micro-controller. There are IDE's and frameworks, such as Quantum Platform [1] and Keil [2], which provide tools to easily generate embedded software for target systems and simulators to debug and test code on host-based environments.

2.3 Web Development

Web development is one of the most popular areas of software engineering. The Internet era requires capable ways of digital communication and transfer of information. Nowadays, the worldwide Internet traffic is massive, with people around the world constantly interacting with each other through the web. It is used for both pleasure and business. Many businesses depend on websites for advertising and, some, are even core based on the web (e.g., e-commerce companies). Hence, the web has become convenient for the society and has turned into an essential part in most people's daily basis. Consequently, web development is a trending topic in the IT world.

Websites have suffered big changes over the years. They are no longer simple static web pages, but complex web applications dealing with huge amounts of information and running on a wide variety of platforms. Web technologies have also evolved, according to the needs, and have introduced new terminologies in software development. It is important to have/use the correct tools in order to implement features and build applications the right way. The demand of creating complex web applications has resulted in web development being sub-divided in

[1] https://www.state-machine.com
[2] http://www.keil.com

two main categories: front-end and back-end. A front-end developer is specialized in the visual interfaces which are displayed for the users. A back-end developer is responsible for all the logic, dealing with processing, validation of data and database interaction in the server-side. The term full-stack is sometimes used and refers to developers who work on both parts of the web application. These advancements have pushed the designation of further new concepts, such as, communication mechanisms between different systems (known as APIs), new software design patterns and the existence of software development frameworks.

2.3.1 Model–View–Controller (MVC)

Model–View–Controller (MVC) is an architectural design pattern widely adopted in web development. It is based on the definition of 3 concepts:

- **Model**: A model is used for interacting with the back-end database. It is an object used for data manipulation on the server side, i.e., selecting, validating and inserting information.
- **View**: A view is a description of how the information is retrieved and displayed to the user.
- **Controller**: A controller is responsible for interacting with the user requests. It sets the available user actions, providing specific methods for data processing and manipulation and the return of views back to the user. It serves as an intermediate between the model and the view.

MVC intends to decouple the different application components, whilst using those 3 layers to separate the logic from the representation (Figure 2.4). Thus, the application's source code is well organized, allowing its maintainability, software scalability and re-usage of independent software components.

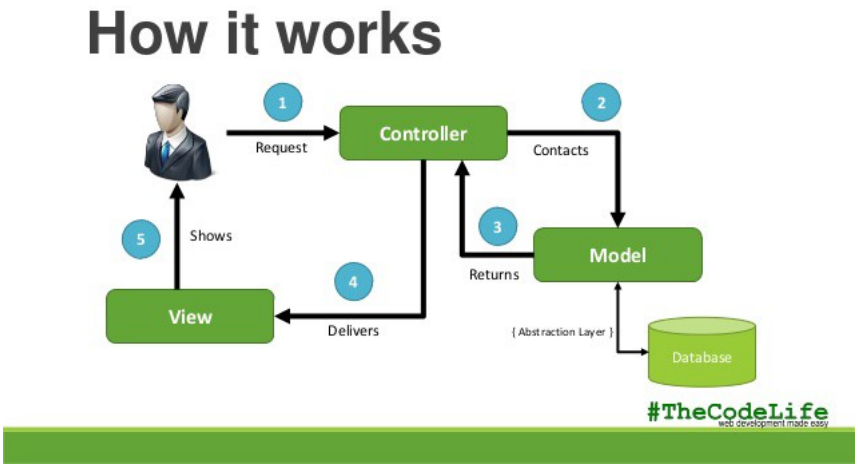


Figure 2.4: MVC Design Pattern (from [47]).

To develop web applications using the MVC model, there are multiple MVC frameworks available for different programming languages. Regarding our project, we implemented our web applications using Laravel, a MVC framework for PHP [39]. Other popular frameworks include Symphony for PHP (from which Laravel is based on), Django for Python and Spring for Java.

Chapter 3

State-of-the-Art

The Environmental Observing Stations we propose are entirely built from off-the-shelf components. The embedded system included in an EOS consists of a main computing platform and all the sensor devices attached. Nowadays, the amount of affordable hardware at our disposal allows us to research and compare several options, choosing the best according to our needs. The construction of our stations is targeted on low-cost budgets, while considering the EOS's purposes, constraints of the deployment location and the system requirements. Depending on the deployment location, an EOS may be restricted in terms of power and/or connectivity. Plus, our software requires the usage of electronic components compatible with certain communication protocols. Thus, it is important to explore the state-of-the-art in terms of hardware available, so we can produce the modular and flexible system we propose. In this chapter, we study some of the off-the-shelf hardware options, focusing more on the prototype we implemented. We address components intended for astronomical and meteorological observing scenarios. We also refer some relevant projects developed for areas similar to ours.

3.1 Computing Platform

An EOS needs a central component responsible for the computational work and the integration of all the external devices. Nowadays, there are available a wide variety of Single Board Computers (SBC). These platforms usually integrate a System on Chip (SoC) with considerable computational power and multiple digital and analog interfaces, allowing the interaction with many external peripherals. They are small electronic components with relatively low power consumption levels. The peculiarity of a SBC is that it supports the usage of operating systems and therefore provides the flexibility to run application software. This ability to run more complex software, in comparison with some more limited micro-controller platforms, designed for embedded software development, is a feature we can take advantage of, at affordable prices. We researched a few of these platforms, according to certain requirements (such as, features, pricing, availability and popularity). The software framework to be used on every EOS is run by the computing platform, which is part of a common backbone structure of hardware. Regardless of the observing goals of

each individual station, the embedded system can be assembled with the same processing unit.

The goal of implementing our prototype was to perform both astronomical and climate monitoring, with each having certain requirements. In terms of astronomy, we wanted to use an astronomy camera for taking sky frames during the night and, possibly, also during the day. So, we wanted a platform which could provide considerable amount of local storage. Regarding meteorology, we wanted to send some of the collected data to Weather Underground, hence needing Internet connectivity. In addition, Internet connectivity was necessary in order to have communication between the EOS and the server and have the complete system functional. Although this constraint is relevant in the choice of a platform, the deployment location of a station can also be a limiting factor.

3.1.1 Arduino

Arduino [5] is a single board with an embedded micro-controller and digital/analog pins to connect sensors and actuators. There are some variants which change the number of pins, the chip or the available RAM or storage for the firmware and program routines. It is a cheap a low-processing power device that offers capabilities to be integrated in a number of projects which require collecting data from the physical world, mechanical automation and event detection upon certain occurrences. It also has available external modules for integration to extend some functionality, such as ethernet shields, to connect the Arduino to the local network and the Internet. Plus, Arduino contains a software IDE and a wide range of libraries that allow to efficiently code programs and flash them to the board's EEPROM. Figure 3.1 illustrates the Arduino Uno model. There are also more Arduino alternatives, such as, the smaller and more compact Arduino Nano, and the Arduino Mega, designed for more complex solutions, offering more pin connections and memory/storage capacity.

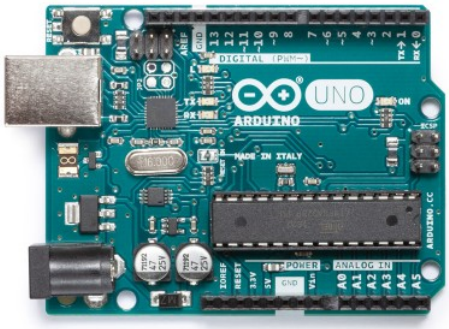


Figure 3.1: Arduino Uno Rev3 (from [7]).

3.1.2 Raspberry Pi

Raspberry Pi [58] is a more powerful device than Arduino, fitted for more complex projects as it is considered a mini general purpose computer (SBC). It is a small circuit board containing an ARM micro-processor and a graphics processing unit, capable of running an operating system (usually Linux). It contains a memory card slot for external storage, USB ports, a network adapter and allows the connection of digital sensors/actuators through its digital pins. Since it offers the generic features of a computer, it is easy to develop software for it and apply it on embedded systems, due to its size and digital interfaces. Figure 3.2 illustrates its most popular and powerful model, Raspberry Pi Model B. There are also available a more affordable model (Raspberry Pi Zero) with smaller size and fewer features and a new refresh of Model B, the Model B+.

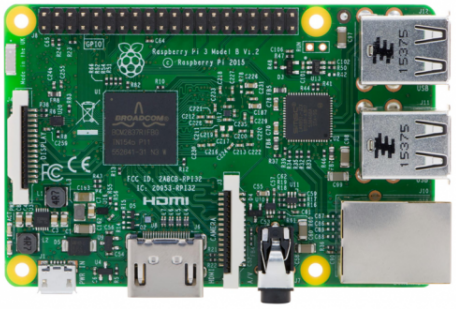


Figure 3.2: Raspberry Pi 3 Model B (from [59]).

3.1.3 BeagleBoard

BeagleBoard [10] is a similar platform to the Raspberry Pi, developed by Texas Instruments. It is an open-hardware/software computer, micro-processor based and can also run Linux operating systems. Some derivatives offer more RAM and computational power comparing to the Raspberry Pi, but are also more costly. Figure 3.3 illustrates the BeagleBoard Black variant.

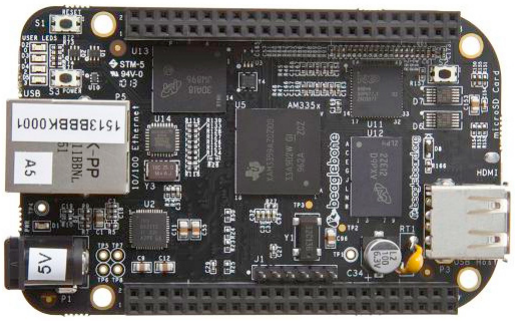


Figure 3.3: BeagleBoard Black (from [11]).

Table 3.1 compares the main characteristics of the three presented platforms.

	Arduino Uno Rev3	Raspberry Pi 3 Model B	BeagleBoard Black RevC
Processor	ATMega 328P	4xBroadcom BCM2837	AM335x
Architecture	AVR	ARM	ARM
Instruction Set	8 bit	64 bit	32 bit
Clock Speed	16 MHz	1.2 GHz	1 GHz
RAM	2 KB	1 GB	512 MB
Storage	32 KB	MicroSD card	4 GB + MicroSD card
Operating Voltage	5 V	5 V	5 V
Analog/Digital Pins	6/14	0/40	7/65
USB Ports	Requires USB-TTL adapter	4 USB 2.0	1 USB 2.0
Network Adapter	Requires external module	Wi-Fi, Ethernet	Wi-Fi, Ethernet
Operating System	N/A	Linux, Windows IoT	Linux, Android

Table 3.1: Table of Processing Platforms.

We have described these three solutions based on defined constraints and hardware availability. There are more alternatives from different manufacturers, at different price ranges, designed for various specific embedded applications.

Our EOS need to integrate a platform with the features required to support our system and run our software. That is, run an operating system, be compatible with addressable communication protocols, provide local storage capacity and have Internet connection. Arduino runs a flashable firmware with programmed routines and Internet connectivity is only possible with a coupled module, which is somewhat limited compared with the built-in alternatives. The Raspberry Pi has a bigger community compared with the BeagleBoard and has better system-designed operating systems, offering more software compatibility and available libraries to function with external components. Additionally, Raspberry Pi can either use a memory card with large capacity or connect USB storage devices supporting considerable storage sizes. Our target necessities make the Raspberry Pi the most affordable and adequate choice for the development of our prototype EOS.

3.2 Sensors

Environmental observation involves sensing a vast range of atmospheric phenomena and, therefore, leans on the utilization of several sensors for the gathering of data.

For each scenario, we establish a kit (a bundle of components) according to the intended purposes and hardware availability and cost. For astronomy, we needed an all-sky camera to record sky phenomena like meteors and aurora borealis and observe transient phenomena. For weather monitoring, the goal was to measure temperature, humidity, wind velocity and direction, precipitation and barometric pressure. We were also interested in measuring intensity of visible and UV light and air quality and pollution. For most of these variables, there are available both analog and digital sensors. Our software framework requires the usage of digital protocols which assign unique addresses to each component. The Raspberry Pi GPIO does not contain analog pins and supports I^2C and One-Wire digital interfaces. Hence, in case of needing to use analog devices, we can pair them with an auxiliary Analog-to-Digital Converter (ADC) [44] (or an external micro-controller). There is also the possibility to convert the receiving data signal to a certain desired protocol, in case the component does not offer default compatibility (e.g., use an I^2C ADC adapter).

3.2.1 Astronomy Cameras

For the astronomy purpose, an astronomical imaging camera is the key component. In a digital camera, the installed sensors are responsible for converting photons into electrical signals in order to reproduce digital images. There are two main technologies of image sensors: CCD and CMOS [16]. Charge Coupled Device (CCD) is the older technology and consists in a two-dimensional array of analog light-sensing elements, each translating into a pixel. Depending on the intensity of the incident light, the elements generate electric voltage which gets converted to binary data through an external ADC. Complementary Metal-Oxide Semiconductor (CMOS) is a more recent technique that has improved over the years and has gradually become an alternative to CCD. CMOS chips incorporate an internal ADC and each element is amplified and converted individually. This results in each pixel being independently converted, instead of all at the same time like in the CCD. CMOS advantages over CCD include lower costs, lower power consumption and lower read-noise levels.

There are several cameras available for astro-imaging from brands like ZWO, QHY and Atik. There are different categories containing models which vary in technical characteristics and in price range. For the implemented prototype, we are using an entry-level ZWO ASI120MM [8].



Figure 3.4: ZWO ASI120MM (from [8]).

Table 3.2 contains detailed specifications of the *ASI120MM* camera. To note that humidity constraints require the camera to be properly isolated inside an enclosure, when operating in exterior environments.

	ZWO ASI120MM
Sensor	MT9M034 CMOS monochrome
Resolution	1280x960 (1.2 Megapixels)
Exposure Range	64 μs to 1000 s
Interface	USB 2.0 (power and data)
Bit rate	12bit ADC
Working Temperature	-5 to 45ºC
Working Relative Humidity	20 to 80%

Table 3.2: Table of specs for ZWO ASI120MM.

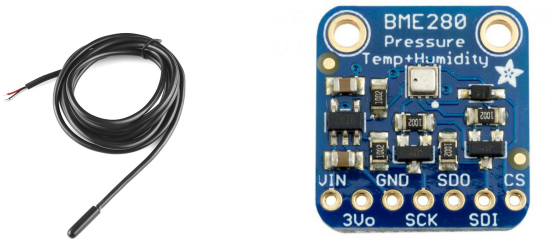
This model is monochrome (no color) and does not contain a cooling system. There are other more powerful models with better quality sensors, communicating through USB 3.0 interface and some produce color images. Some integrate a cooling system and require external power supply, e.g., ZWO ASI174MC [1]. ZWO company maintains a Software Development Kit (SDK) [9] to allow its cameras to be used in software development projects, such as ours.

3.2.2 Temperature and Humidity

The monitoring of weather conditions typically involves a multiplicity of sensors. In terms of temperature and humidity, usually they are both integrated in a single digital sensor, existing several simple and low-cost options [12, 22, 45, 66]. In order for it to work properly in the exterior, it should be protected, whilst providing adequate conditions for its correct functioning [33, 83], e.g., the sensor cannot be located too close to the mainboard as it affects temperature

[1] https://astronomy-imaging-camera.com/product/asi174mc

readings. The waterproof DS18B20 [27] is a digital sensor (One-Wire interface) with the wires covered in PVC, so it can be placed in strategic spots in the exterior.



(a) DS18B20 (from [27]).

(b) BME280 (from [13]).

Figure 3.5: Humidity/Temperature sensors.

Table 3.3 compares the technical specifications of some humidity/temperature sensors suitable for our work.

Sensor	Hum. range	Temp. range	Pins	Type
DHT22	0 to 100% ±2 to 5%	−40 to 80ºC ± 0.5ºC	4	Digital
SHT1x	0 to 100% ± 2 to 4.5%	−40 to 123.8ºC ± 0.3 to 0.5ºC	4	Digital
DS18B20	N/A	−55 to 125ºC (± 0.5ºC from −10 to 85ºC)	3	Digital (One-Wire)
BME280	0 to 100% ±3%	−40 to 85ºC	4	Digital (I^2C/SPI)
MCP9808	N/A	−40 to 125ºC ± 0.25ºC	board	Digital (I^2C)

Table 3.3: Table of Humidity/Temperature sensors.

3.2.3 Wind Speed and Direction

For the wind, there are anemometers to measure the speed and wind vanes to measure the direction. As for the anemometers, there are the common mechanical ones, usually in cup format or propellers, that use electric switches to measure the speed from the rotation of the cups. The accuracy of this type of sensor can be affected under heavy precipitation conditions. Alternatively to mechanical anemometers, ultrasonic anemometers offer a great step-up in precision, but also

in cost. An ultrasonic anemometer sends and receives sound waves and calculates the speed of the wind by the delay time between sending and receiving. For the wind direction, windsocks and wind vanes can be used. Pre-built and easy-to-use modules [3] can be purchased or custom-made wind vanes can be assembled, integrating the necessary sensors.



Figure 3.6: Anemometers (left to right: ultrasonic, cup and wind vane) (from [4]).

3.2.4 Rain Intensity

For measuring rainfall intensity, the simple way consists in manually determining the rainfall amount from a collecting vessel over a period of time. In order to get continuous monitoring and get more reliable readings at more intense precipitation conditions, the bucket should have a self-emptying mechanism, like tilting or pivoting [33]. A digital rain gauge integrates a magnetic reed switch and a digital counter. The counter increments each time the switch is triggered and the result is the rainfall amount relative to the bucket's volume.

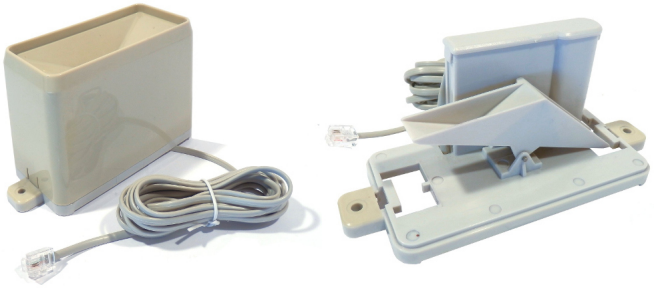
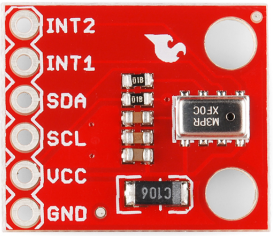


Figure 3.7: Raspberry Pi Rain Gauge Sensor (from [26]).

3.2.5 Atmospheric Pressure

Atmospheric pressure can be measured with low-cost and precise digital barometric sensors [12, 24, 46]. From the absolute values measured, we can calculate the sea-level pressure based on the altitude of the location. Some of these hardware modules also have built-in temperature sensors.



(a) MPL3115A2 (from [46]).



(b) SM1131 (from [24]).

Figure 3.8: Digital barometric air pressure sensors.

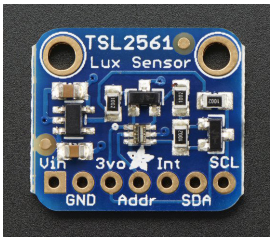
Table 3.4 shows the main attributes of each of the two digital sensors.

	MPL3115A2	SM1131	BME280
Pressure type	Absolute	Absolute	Absolute
Pressure Range	20 to 110 kPa	60 to 165 kPa	300 to 1100 hPa
Accuracy	±0.1 kPa	±1.0 kPa	±0.2 Pa
Digital Interface	I^2C	I^2C, SPI	I^2C, SPI
Temperature range	−40 to 85ºC	−40 to 125ºC	−40 to 85ºC

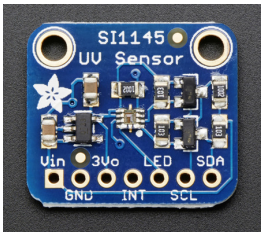
Table 3.4: Table of digital barometric air pressure sensors.

3.2.6 Luminosity

Regarding luminosity, there are available analog and digital sensors [75] capable of measuring infrared and/or human-visible spectrum. These sensors determine light levels due to its integrated photo-resistors, which decrease their resistance when they are exposed to light. For UV light, there are low-cost digital [25] and analog sensors as well. In order to use an analog sensor through a digital interface, we can pair it with an ADC or a capacitor, in case it is the only analog sensor connected.



(a) TSL2561 Luminosity sensor (from [75]).



(b) SI1145 UV light sensor (from [25]).

Figure 3.9: Luminosity sensors.

Table 3.5 shows the characteristics of the digital luminosity sensor TSL2561 and the UV sensor SI1145. The SI1145 also contains an embedded infrared proximity detector.

	TSL2561	SI1145
IR spectrum wavelength	550 to 1000 nm	550 to 1000 nm
Visible spectrum wavelength	300 to 1000 nm	400 to 800 nm
Dynamic Range	0.1 to 40 000 lx	0.1 to 40 000 lx
Digital Interface	I^2C	I^2C

Table 3.5: Table of luminosity sensors.

3.2.7 Air Quality

We also considered the possibility of monitoring general air quality (contaminants, such as atmospheric aerosol particles, ozone, nitrogen dioxide and carbon monoxide gases). There are sensors to evaluate concentration of specific gases [74] and, for instance, a more costly sensor detection framework (GAS 86xyB [65]), which allows simultaneous reads of multiple compounds.

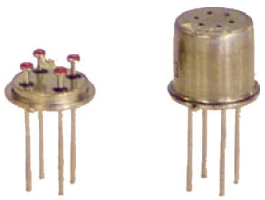


Figure 3.10: TGS 2600 Gas sensor (from [74]).

3.2.8 Position Tracking

On certain scenarios, it might be interesting to track the station's location. For that, there are GPS sensors, such as the one illustrated in Figure 3.11. Relevant to add that this kind of devices drain a considerable amount of power and might not be appropriate to use if the station is operating on battery mode.

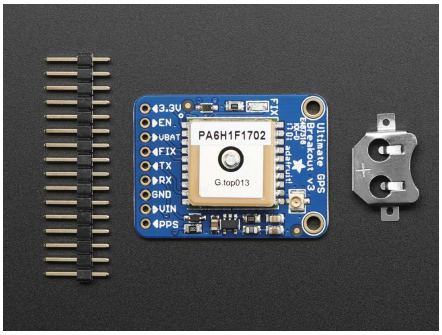


Figure 3.11: MTK3339 GPS module (from [30]).

3.3 Power Availability

In some scenarios, an EOS must function on locations with no electrical supply. This is made possible through the use of batteries or, as they are most often referred as, powerbanks. Modern powerbanks use Lithium-ion (e.g., 18650 batteries) or Lithium-polymer battery cells. Li-polymer powerbanks are more reliable and offer more stability due to its integrated protection circuit against overcharges and short-circuits. Usually, Li-Ion power-banks are cheaper and do not incorporate protection circuits [56].

Since the Raspberry Pi operates on DC current through an USB port, the use of powerbanks offers a solid solution for temporary applications. The available capacities (measured in Ampere per hour) typically range from 5Ah to 20Ah and output through a USB port at 5V.



(a) Xiaomi 2C li-polymer powerbank 20000mAh (from [54]).



(b) Poweradd Apollo solar panel powerbank 7200mAh (from [55]).

Figure 3.12: Example of li-polymer and solar panel powerbanks.

For longer lasting running times, solar panels offer a self-sustaining power source. There are powerbanks with integrated solar cells at a accessible prices (Figure 3.12). The assemble of a photo-voltaic module system requires a set of equipment. Because it outputs DC current at 12/24/48 volts, we need to use a DC regulator (charge controller) between the solar panel and the powerbank to offer steady 5V. Figure 3.13 illustrates a DC regulator with USB output. If AC current is required, we will also need an inverter (Figure 3.14).



Figure 3.13: Example of a USB voltage step-down regulator (from [19]).

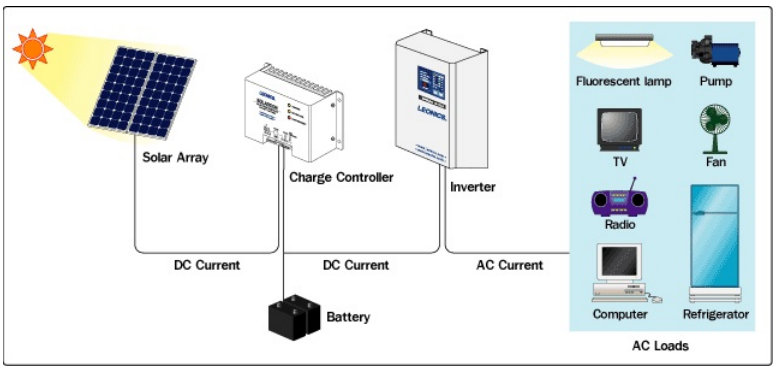


Figure 3.14: Solar powering system for DC and AC devices (from [70]).

In order to protect the Raspberry Pi and other attached electronic components against power outages and power surges, we can use an Uninterruptible Power Supply (UPS). When the main power fails, an UPS provides power to connected devices for a short amount of time. In that period, the devices can be properly shutdown - an important feature for the Raspberry Pi, for example, to prevent SD card corruption. UPS PIco [77] is an Uninterruptible Power Supply for the Raspberry Pi models (Figure 3.15). It is equipped with a lithium polymer battery and provides over current/voltage protection with 5V 3A output power.

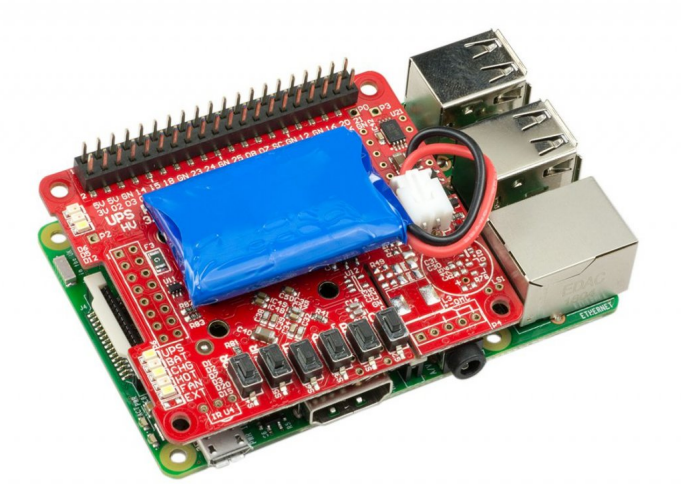


Figure 3.15: UPS PIco - Uninterruptible Power Supply (HV3.0B+) (from [77]).

We can also adopt power management techniques such as powering the devices/sensors only on scheduled periods and limit the computational processing depending on the powering conditions. Furthermore, we could use a GSM relay switch (Figure 3.16) so it can be possible to remotely power up and make the station operational with a simple phone call or SMS.



Figure 3.16: GSM relay switch (from [32]).

3.4 Connectivity and Communication

An EOS requires communication mechanisms in order to allow remote control and be able to send collected data to the server and/or external services. This communication can be constrained by the deployment location. The Raspberry Pi has an ethernet port and an integrated Wi-Fi adapter that is sufficient for locations with Wi-Fi connectivity. However, if the station is to be deployed in a remote region without Wi-Fi coverage, it becomes ineffective. Another possibility is using cellular networks (using 3G/4G modules), although the network coverage can still be limited.

A more power efficient option is the use of Low Power Wide Area Network (LPWAN) communication technologies [31, 85]. These technologies are designed for long range communications at low bit rates and, therefore, are intended for devices with power constraints (e.g., battery powered). There are several technology standards for LPWAN: the family of Narrowband-IoT standards proposed by 3GPP and proprietary solutions like LoRa and Sigfox. The narrowband radio technologies standardized by 3GPP include LTE-M [42] (over 4G networks) and EC-GSM-IoT [28] (over GSM networks) and operate in licensed radio frequencies. Several companies provide other LPWAN solutions, including u-blox [2], Sigfox [3], Libelium [4], Weightless [5] and LoRa Alliance [6]. These technologies operate in ISM (Industrial, Scientific and Medical) radio band.

Sigfox implements a lightweight communication protocol using ISM radio frequencies with small payload data packets, specifically designed for battery powered IoT devices (Figure 3.17). Configuring the connectivity to the Sigfox network involves the usage of a specific module, which can be bought or assembled with the necessary parts (Figure 3.18). This module integrates its own micro-controller and an RF transceiver and can easily be set up using Sigfox API. The Internet access is granted by the wide coverage network maintained by several Sigfox operators throughout the world (e.g., Narrownet [7] in Portugal). In the Sigfox network, the end-nodes (the devices connecting the modules) communicate through radio with the Sigfox base-stations, which in turn communicate through IP connections with the Sigfox cloud infrastructure, serving as a gateway to the Internet.

[2] https://www.u-blox.com/en/solution/technology/narrowband-iot-nb-iot
[3] https://www.sigfox.com/en/sigfox-iot-technology-overview
[4] http://www.libelium.com
[5] https://www.ubiik.com/lpwan-technology
[6] https://lora-alliance.org/about-lora-alliance
[7] http://narrownet.pt

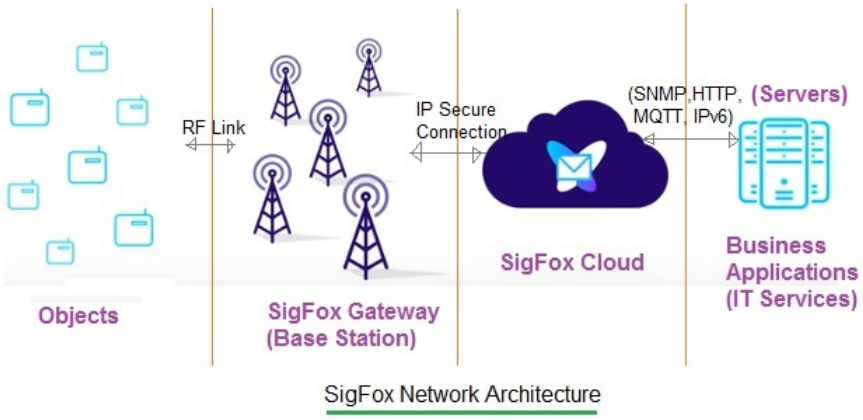


Figure 3.17: Sigfox network architecture (from [67]).



Figure 3.18: Sigfox wireless shields (from [68]).

LoRa is a LPWAN spread spectrum technology (it uses Chirp Spread Spectrum). It is intended for wireless devices in the Internet of Things world, offering connectivity to a long range global network (LoRaWAN). To use this technology, we need to integrate a LoRa module (Figure 3.19). We can also build a LoRa gateway, extending the network range, or connect to an existing one. The nodes and the gateway communicate through different frequency channels and the gateways communicate with the central network infrastructure over standard IP connections. This technology is also intended for long range communications with small payload packets. LoRa also offers the ability to set a private network communicating only with specified services.



Figure 3.19: LoRa wireless shields (from [40]).

Table 3.6 compares the technical specifications of technologies NarrowBand-IoT, LoRa and LTE-M [41, 42, 69].

	LTE Cat NB1 (NB-IoT)	LoRa	LTE Cat M1 (LTE-M/eMTC)
Spectrum	Licensed	Unlicensed	Licensed
Modulation	QPSK	CSS	OFDMA, QAM
Bandwidth	180 KHz	500 Hz - 125 KHz	1 MHz
Download Peak Data Rate	235 bps	290 bps	1 Mbps
Upload Peak Data Rate	205 bps	50 bps	1 Mbps
Transmission Power	23 dBm	23 dBm	20 dBm
Link budget	150 dB	154 dB	160 dB
Mobility	No connected mobility (only idle mode reselection)	Mobility support	Mobility support

Table 3.6: Table of LPWAN technologies.

In terms of required hardware, we can use embedded cellular modems, such as Digi XBee (Figure 3.20). There are several models available for GSM, 3G, LTE and LPWAN. These modules integrate API's for communicating with the device by software.

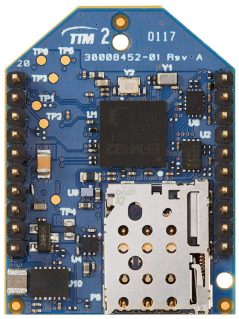
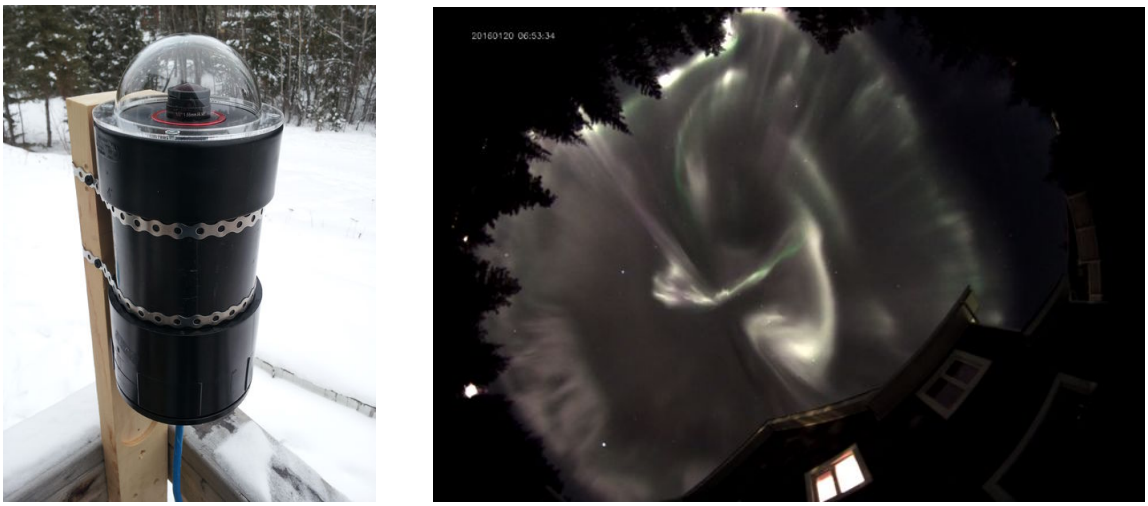


Figure 3.20: Digi XBee Cellular NB-IOT (from [23]).

3.5 Available Solutions

3.5.1 Astronomy

Project [37] is a system for gathering images of night sky. Thomas Jacquin, the project creator, integrated a ZWO ASI224MC camera on a Raspberry Pi 3, building a homemade external enclosure for the material's isolation. The software used was a modified script example, distributed by ZWO, using ASI SDK to communicate with the camera and compiled for systems with ARM architecture. The source code is available on GitHub [29] and also served as base for our development with the ASI120MM camera.



(a) Project enclosure.

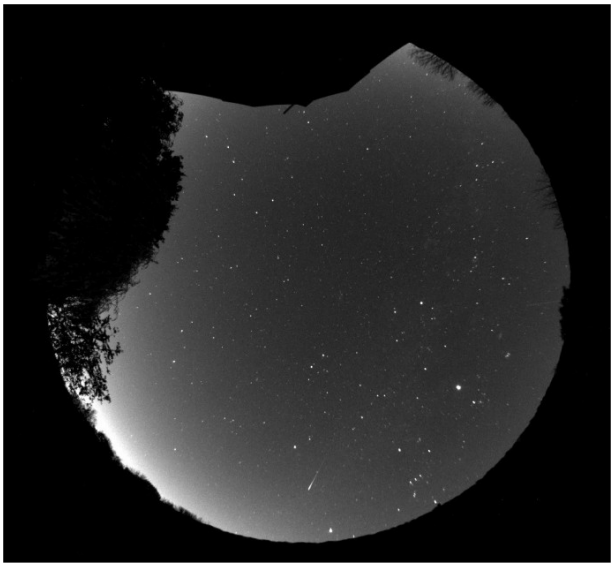
(b) Aurora borealis captured in Canada.

Figure 3.21: Images from project [37].

The Oculus all-sky camera [48] is a professional astronomical camera, developed by Starlight Xpress Ltd and is designed to provide high quality sky images for weather and meteor observations (Figure 3.22). It features a top-class Fish Eye lens with 150 or 180 degrees of coverage and costs around 1000€. It is a "next level" camera to be applied on professional application fields, e.g. public astronomical observatories, in which they expect high level of image detail and infinity focus to observe planets and distant constellations.



(a) Model structure and enclosure.



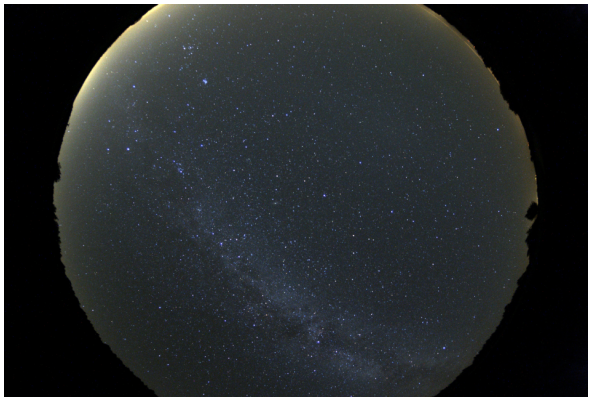
(b) Capture example taken with Oculus.

Figure 3.22: Oculus all-sky camera [48].

The OMEA all-sky camera [49], analogously to Oculus, is a top-class astronomical camera, developed by Alcor System, and its purpose is to record both day and night sky events. Due to its high sensitivity, it is capable of capturing all sorts of phenomena, including meteors and fireballs. There are a few models available, varying from a monochrome or colored sensor, higher resolutions and wider fields of view. All OMEA cameras have embedded a cooling system. There is also accessible for purchase, a weather station module to attach to the camera (Figure 3.23). It includes sensors to measure weather variables, such as temperature, humidity and air pressure. It also contains a wind vane and a cup format anemometer to calculate wind direction and speed, respectively. Alcor System currently has several systems running worldwide and displaying real-time imaging [8].



(a) OMEA camera with weather station module.



(b) Taken with 6C model. Settings: exposure-time=35s

Figure 3.23: OMEA system [49].

[8] http://www.astrosurf.com/cavadore/meteo/current/allSkys.html

3.5.2 Meteorology

Most of the weather stations reporting data from around the world are commercial solutions. These are products which include a bundle of common sensor components and, usually, run proprietary software. A popular manufacturer of commercial weather stations is Davis Instruments [9]. The company specializes in providing quality equipment for weather and crop monitoring. Their solutions integrate a solar panel for powering the station and a console device, which is responsible for receiving the collected data and displaying it on the screen (Figure 3.24).



(a) Davis Vantage VUE (from [18]) (b) Davis Vantage PRO2 (from [17])

Figure 3.24: Davis Instruments Solutions.

There is also available hardware for building personal weather stations. Raspberry Pi organization supplies a Weather Station kit [60], composed by a single circuit board to connect to a Raspberry Pi (Figure 3.25). This board contains sensors to monitor usual environmental variables, i.e., temperature, humidity and air pressure. It also contains ports to connect external rain gauges, anemometers and wind-vanes.

There are guides for building weather stations using off-the-shelf components [6, 61], including how to assemble normal cup/propeller anemometers and a more sophisticated ultrasonic anemometer [76].

[9] https://www.davisinstruments.com

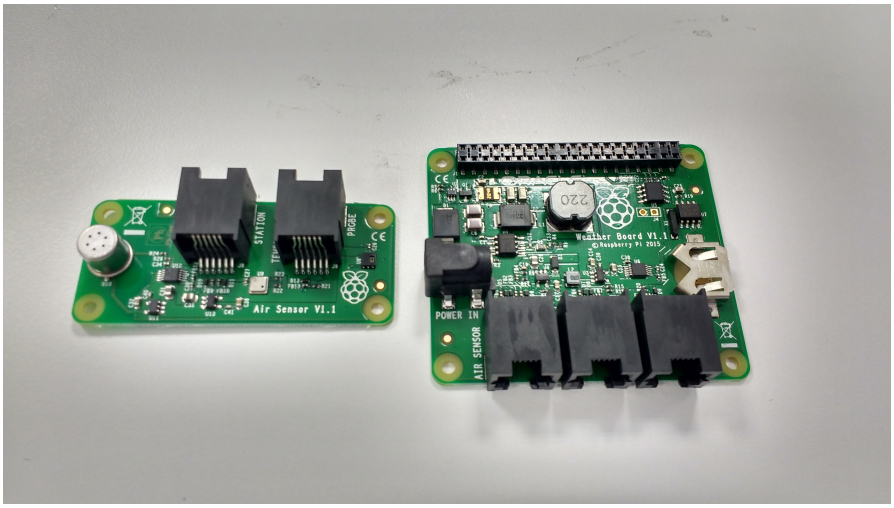


Figure 3.25: Raspberry Pi Air Sensor Board and Weather Board (from [60]).

AirPi [2] is a project which developed a kit, currently discontinued, with a set of sensors (temperature, humidity, air pressure, light levels, UV levels, carbon monoxide, nitrogen dioxide and smoke level) and an Internet module with configurable software to send collected data to a specific remote location on the Internet. On its website there is also a description on how to implement a shield module, listing the electronics required and some simple sensors available for measuring weather variables.

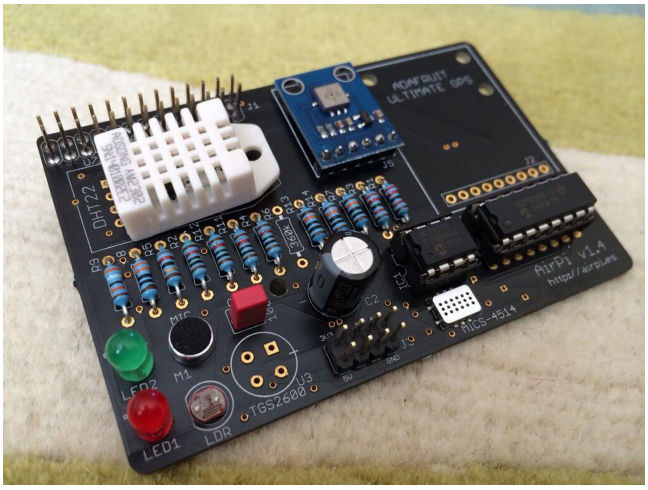


Figure 3.26: AirPi (from [2]).

The Weather Board [82] is a weather station board, developed by SwitchDoc Labs, designed to be integrated on a Raspberry Pi or Arduino with the corresponding software available online for download.

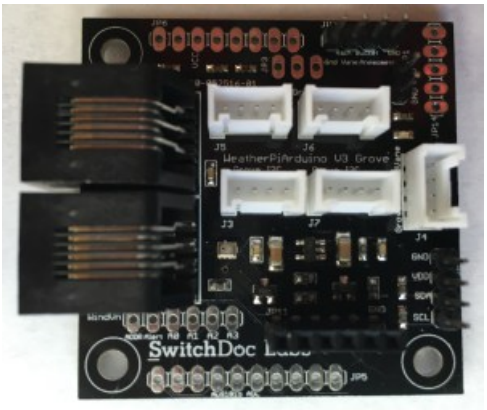


Figure 3.27: SwitchDoc Weather Board (from [82]).

The weather station presented by Popa & Iapa [53] consists in a low-cost embedded system which monitors temperature, humidity, pressure and luminosity using a GSM module to retrieve information and receive control commands through SMS.

Sankar & Norman designed a mobile weather monitoring system, presented in the literature [63]. The station consists in a hydrogen-filled balloon that measures temperature, humidity and barometric pressure, while moving. This moving system transmits the measured data to a ground station by radio frequencies.

Chapter 4

Architecture

In this chapter, we describe the architecture of an EOS, its basic components and interactions.

4.1 Requirements

We established multiple requirements previously to the design and implementation of our system. We wanted a versatile way of performing environmental observation. That is, develop a system with the ability to combine several deployed stations (EOS) for particular environmental scenarios. Furthermore, the building process of an Environmental Observing Station should be modular, in the sense that it's customization would not be too complex for non-specialist users. Each EOS should be able to work independently, but with the possibility of synchronization with a central entity.

In terms of mandatory features, each EOS must provide mechanisms of configuration of its devices as well as capturing data. Additionally, it can be tasked to upload some of that data to external services (e.g., our prototype sends meteorological information to Weather Underground). There are also power and connectivity constraints associated with each EOS. Those factors can be limited by the deployment location (e.g., possibility of no electric connection or Internet access nearby). For instance, Internet connection might be necessary for accessing EOS's services from the Internet or in case of needing to communicate with the server. Thus, for each deployment, we have to analyze the scenario and configure the EOS accordingly; power constraints can be solved by powering it with batteries (and possibly solar panels) and communication can be made over cellular/LPWAN, instead of IEEE 802 standards.

After an EOS is assembled, configured and deployed, it must be able to operate without any user intervention. That is, it must perform auto-detection of all the connected components and run the capturing software with pre-set configurations. Moreover, a user should be able to change configuration parameters and activate/deactivate components with ease. We refer to a detected device as a device being connected to a computing platform and successfully recognized and mapped to its corresponding configuration in the database. A detected device is a component

ready to be used by the capturing software.

Our software must support any component using USB, One-Wire or I^2C interfaces. Due to One-Wire/I^2C addressing system, the detection process is done in a straight-forward manner. Regarding USB, it may require some software tweaks beforehand. The running software makes use of a local database, stored individually on each EOS. In order to run the software, each EOS must integrate a computing board capable of running Linux (e.g., Raspberry Pi).

The ability to view captured data is another requirement. We wanted to somehow process the raw data that is captured and transform it into meaningful information, which could be displayed for users. For that, all the captured data is submitted to a normalization process in order to give flexibility of fetching and organizing the information in different ways and support future data analysis. For our implemented use case, it is important to show information as time series. We opted for a centralized approach for both storage and display of data from each deployed EOS. Thus, our system includes a server responsible for retrieving, processing and displaying that information.

4.2 Overview

Our platform has as core components a centralized server and a number of deployed stations (Figure 4.1).

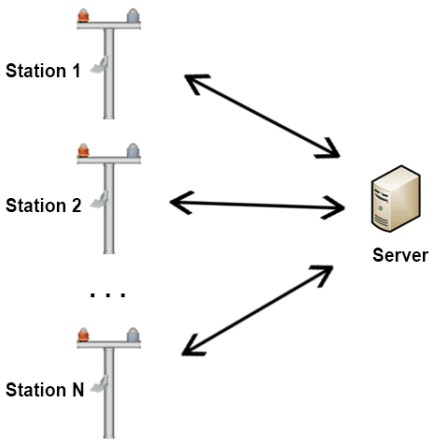


Figure 4.1: Diagram of components of the global system.

We have a centralized server for storing and managing information related to all deployed stations. The server is responsible for retrieving data from the EOS and presenting it as visual content to users. It also serves as a node of synchronization of information for all stations. The EOSs and the server communicate through a protocol of structured messages (API). Figure 4.2 illustrates a diagram of the software components implemented in the centralized server.

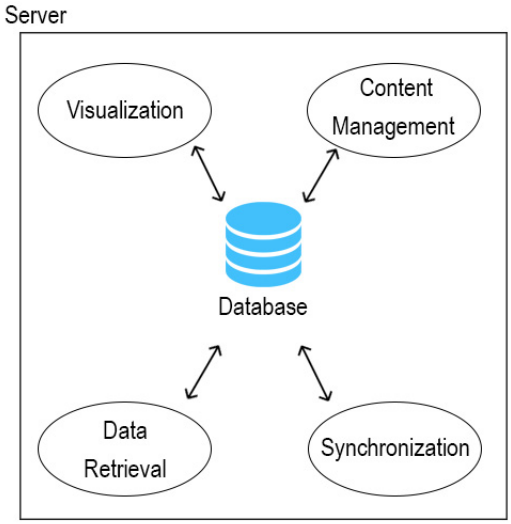


Figure 4.2: Diagram of software components of the server.

Every deployed EOS performs environmental observation for a specific scenario. While operating, an EOS allows a customization of its connected components. Every device can be configured with usage parameters within its specification. Plus, the stations run a capturing software framework, which operates according to pre-set settings. Both the configuration and capturing processes use an auto-detection mechanism in order to identify plugged components and determine if they are ready to be used. Additionally, each station can require a database synchronization with the server. It is a feature that requires available connectivity to the server, but it keeps the station up-to-date with information related to all devices compatible with the system. Thus, new devices can be added and used by that EOS. Figure 4.3 illustrates a diagram of the software components present in every EOS.

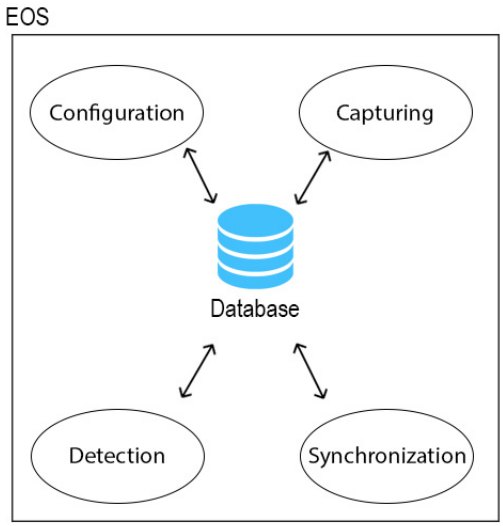


Figure 4.3: Diagram of software components of an EOS.

Every station contains an embedded system, besides its physical structure. The embedded system consists of a computing platform connected to various components (Figure 4.4). The computing platform is the main component, interconnecting all the other devices and running all the software. We consider a device as a piece of equipment (e.g., temperature sensor, digital camera, anemometer). A device is a component which can be connected to a station. During this thesis, we use the concepts of device and component interchangeably. A device can be of several types: processing (i.e., a board with a micro-controller, memory and interfaces), communication (e.g., network adapters), power (e.g., powerbanks, solar panels) and sensoring (e.g., a device for sensoring temperature, motion detector). A sensor is a module within a device which is responsible for harvesting information from the physical environment. A device may contain several sensors (e.g., the device BME280 contains 3 sensors: temperature, humidity and pressure).

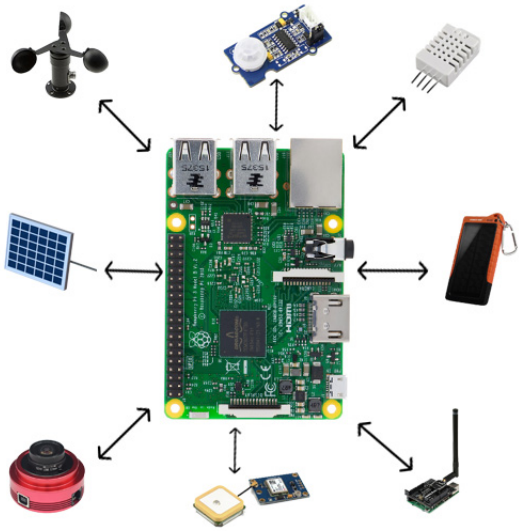


Figure 4.4: Diagram of hardware components of an EOS.

Henceforth, we sometimes apply the term *kit*, referring to a station (EOS). We assume a kit as a collection of certain devices assigned for a particular environmental scenario, thus an EOS.

The server communicates with each EOS to synchronize configuration parameters and to retrieve captured data. After a station is assembled and configured with a base system configuration, it is able to operate autonomously without communication to the main server (Internet connectivity may not be possible). However, it will not be able to request synchronization and the server will not display updated information of the given station. The EOS users will still be able to use the station and configure its local components from the local network.

The web applications developed in this project were designed to separate the presentation of content to the user (known as front-end) and all the logic and data processing (known as back-end). They were implemented using Laravel, a MVC framework. We have also created RESTful APIs to allow other systems to communication with the back-end system, as is the case of our Android app for the configuration of an EOS.

4.3 Server

The server has three main purposes:

- Store a database for all the known devices and deployed stations;
- Retrieve and process data dumps from the stations;
- Provide visual content display for each station.

The server runs a web server with two applications. An application for managing content of the database ("Kit Insert" app) and another for displaying updated information from each deployed EOS ("Kit View" app). The application's conceptual models are illustrated in Figure 4.5 and Figure 4.6, respectively.

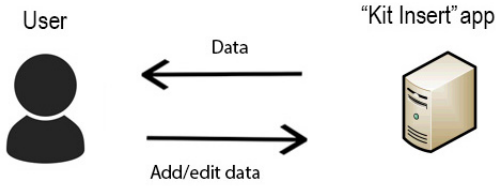


Figure 4.5: "Kit Insert" Conceptual Diagram.

"Kit Insert" application is for managing content in the central database. The data displayed to the user includes information about devices (e.g., specifications, types of collected data and default usage configurations) and deployed EOS. Authenticated users can also edit content or add new data.

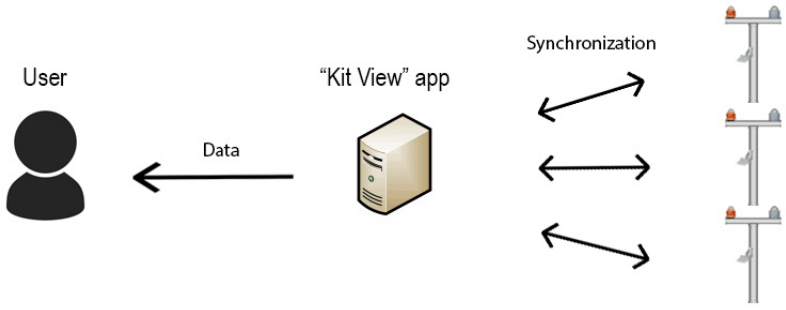


Figure 4.6: "Kit View" Conceptual Diagram.

"Kit View" application is for providing visual content for each station. The server performs data synchronization with each online EOS, i.e., retrieves the captured data and inserts it in the database after validating it. The server can also execute a data cleansing mechanism in which

aged data is removed from the database, as well as, data dumps deleted from an EOS. The app displays data organized per EOS: time series charts (e.g., for a temperature component) and most recent pictures and/or timelapse videos (e.g., for a camera).

Both applications interact directly with the local database. The users can access those applications from the Internet through a web browser (Figure 4.7).

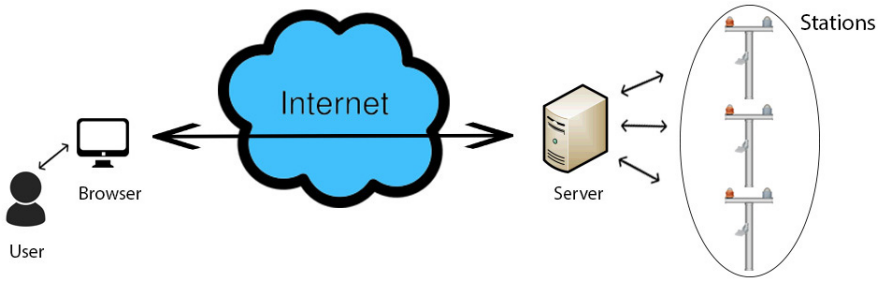


Figure 4.7: Diagram of communication between users and the server.

"Kit Insert" app implements several features (Figure 4.8). The user interacts with a controller, which allows multiple types of requests. The user can request to list available content, edit certain values and insert new records (e.g., add information of a new EOS to be deployed). The controller has defined actions for each specific request. The listing process involves retrieving data from the database and providing it to the client. Thus, the user is able to see relevant content stored in the server. Specifically, information about devices, their default usage configurations, their associated data types and deployed stations. The user can also edit or insert content through the application's forms (which requires authentication). The server is responsible for processing the submitted input, performing a correct validation and committing it to the database, if successful. The back-end system also has a RESTful web service for allowing external systems to request content.

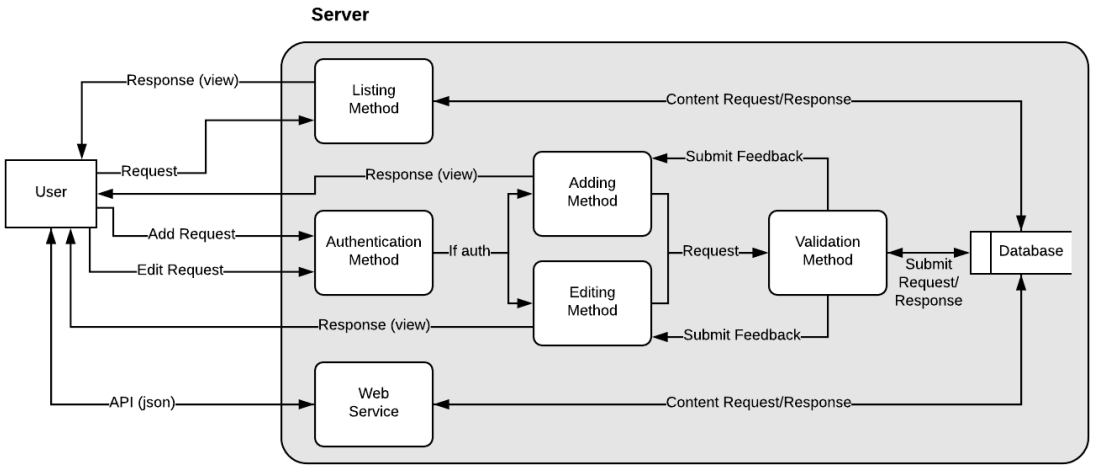


Figure 4.8: "Kit Insert" Data Flow Diagram.

"Kit View" app provides visualization of captured content, per EOS, and also performs data retrieval and cleansing operations (Figure 4.9). An authenticated user can also request the server to synchronize with the deployed EOS, displaying updated information. The synchronization with each station includes several procedures. First, the retrieval of dump data (incremental transfers) with possible data removal from the source (if requested, in order to keep the EOS with free available storage). Then, the transferred data is submitted to processing (token parsing, validation and database insertion). The communication between the "Kit View" back-end server and the EOS can be done either over encrypted connections (using SSH with public key authentication) or by HTTP requests (HTTPS can also be used, depending on infra-structure settings and the type of data to be transferred). The app also involves a process of data cleansing. It can either be scheduled to run periodically or as an on-demand task. Its purpose is to remove older records from the database in order to keep it from reaching extensive sizes. As an example, if data is being captured at fast cadences (e.g., every ten minutes) for several months, it can translate into several thousands of stored records. Therefore, by removing older information, the database access is not compromised and the applications do not suffer from performance loss.

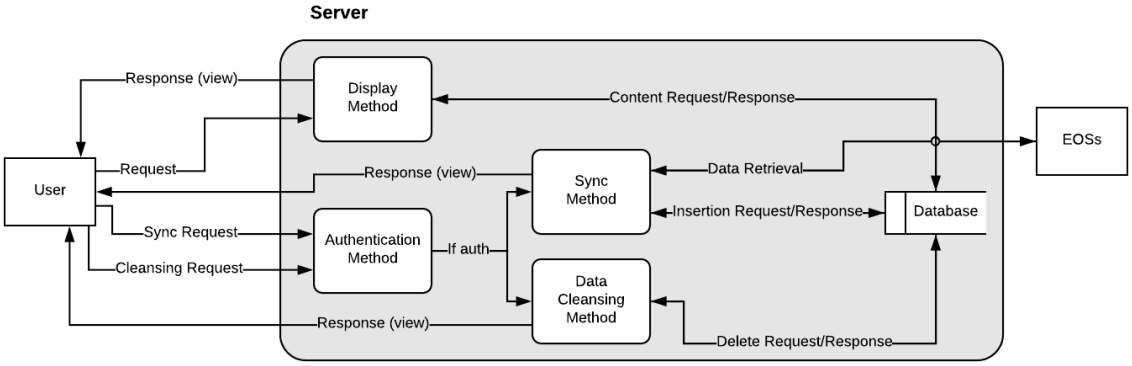


Figure 4.9: "Kit View" Data Flow Diagram.

4.4 Station

Each station works independently, performing an automatic recognition of connected devices and mapping their usage configuration from a local database. Having connectivity to the remote server and in case of needing to attach a new device, each station can synchronize its local database from the central one, adding new devices and types previously inserted in the server. Each station runs a local web application ("Kit Config") for the configuration of all its connected and recognized devices, as well as, the ability to activate/deactivate the whole station or a particular device. These operations require an authenticated user. The application's conceptual model is illustrated in Figure 4.10. All the configuration parameters uploaded to the server are validated before being updated in the database. The application also implements a RESTful API, used by the Android app for getting the configurations and for submitting their changes.

The app data flow diagram is displayed in Figure 4.11.

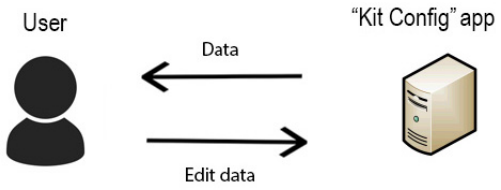


Figure 4.10: "Kit Config" Conceptual Diagram.

Through the "Kit Config" application, the user is able to see all the recognized devices by the system. Authenticated users can configure each device (e.g., activate/deactivate the component and edit usage parameters). All usage properties applied by the user are validated according to specifications defined for each device.

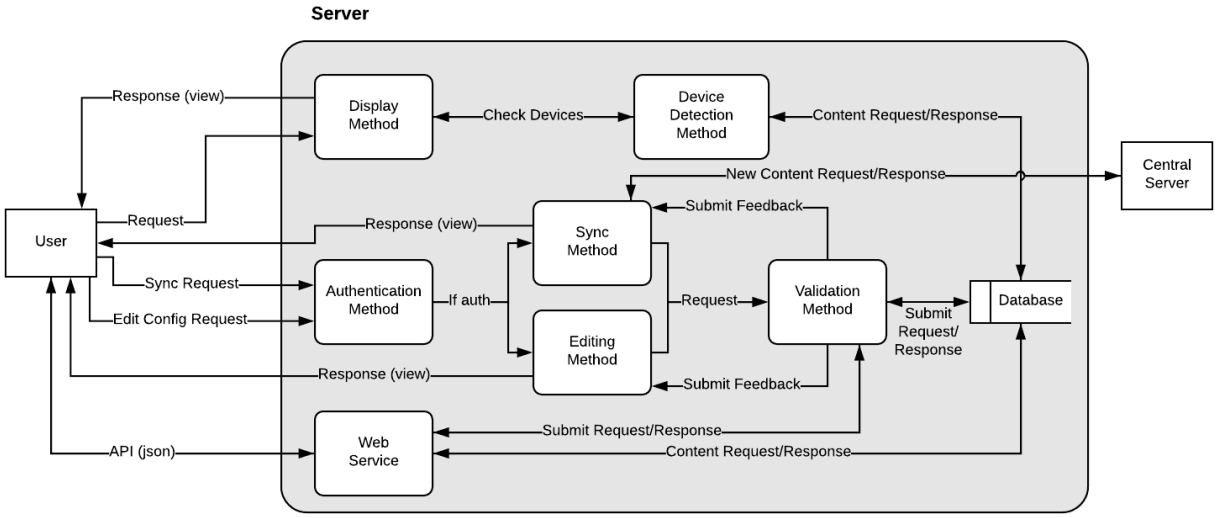


Figure 4.11: "Kit Config" Data Flow Diagram.

The execution mechanism of the capturing software is illustrated in Figure 4.12. The program can be initiated by a user or as a scheduled task. It first checks it the station is online, by verifying its activation parameter in the database. The program exits if the station is offline. Otherwise, it moves to device detection. Having plugged components, their address is matched to the recognized devices in the database. If any matched, a capturing process is started for each recognized device. Afterwards, the program runs indefinitely until a kill signal is received.

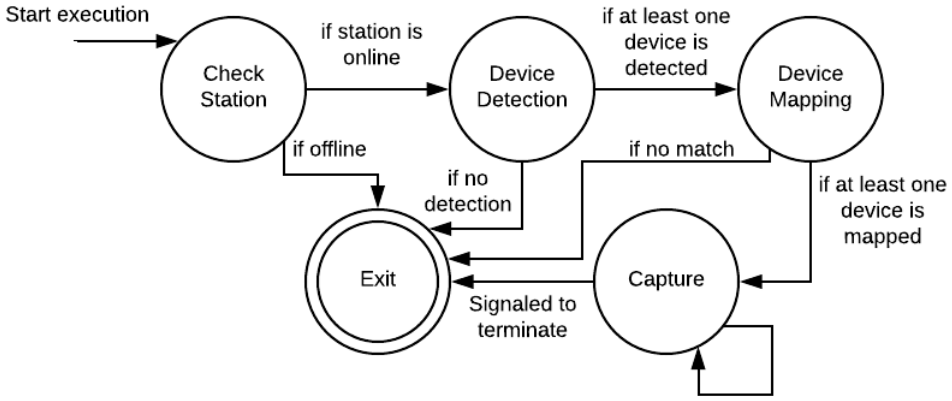


Figure 4.12: Capturing Software State Machine Diagram.

We opted for a centralized architecture for several reasons. The separate server acts as "master" for all deployed stations, allowing to have a system with multiple EOS deployed. Thus, it provides an overall perspective of active stations from a single point. Furthermore, the server alleviates the EOS from executing certain tasks. It is an useful feature in order to limit workload, local storage usage and power consumption in each station. Since an EOS is intended to operate in an outdoor environment and, thus can easily overheat due to the sun, it is important to maintain the system's temperature within acceptable ranges to prevent thermal throttling and possibly damage of electronic components. Depending on the complexity and scalabity of the system, having a central entity responsible for providing services to the other nodes has its benefits.

Nonetheless, the server could perfectly be integrated in an EOS. It would be a feasible approach, particularly if the goal was to only have a single station deployed. A station with no power consumption constraints and which does not require heavy data processing, can operate as an autonomous system. Computing platforms such as the Raspberry Pi have the necessary hardware resources to allow the implementation of such a system.

4.5 Concrete Example

In order to clarify the comprehension of how the system works, we present below a practical scenario.

As assumption, we intend to assemble an Environmental Observing Station for meteorological purposes. That is, we aim to harvest typical weather variables (e.g., temperature, humidity, atmospheric pressure, precipitation and wind speed/direction). For such, we need to determine which sensor devices are to be used. Additionally, a computing platform is required in order to integrate all the external devices and run the software. The most relevant features to look

for when choosing the platform are interfaces for connecting external peripherals, Internet connectivity options and operating systems support. The software performs auto-detection of plugged devices, and therefore requires the devices to communicate through digital protocols with an address allocation system (e.g., I^2C). Thus, both the platform and the sensors need to be able to operate on such interfaces. The system should also have Internet access in order to communicate with the server. So, the platform must support connectivity technologies (e.g., IEEE 802.11, 4G, LPWAN). The deployment location of the station can also establish some constraints, specifically in terms of Internet connectivity and power delivery. Isolated locations can require the usage of long range communication technologies and lack of electricity can affect how an EOS is powered, demanding the utilization of alternative powering solutions.

Assembling an EOS also involves building a physical structure. It serves as shelter for the embedded system, while providing a correct positioning for all the sensor components. Its construction can be based on off-the-shelf components, such as the prototype we assembled, or a commercial product. The benefit of a custom-made physical structure is that its design is based on the devices to be used, and thus it is shaped accordingly. When building a homemade physical structure, we need to keep in mind the type of materials used. The usage of non-corrosive metals (e.g., aluminum, stainless steel or galvanized steel) provide the durability required for long-lasting outdoor usage, without the need for continual maintenance. Also, metal materials offer more stability compared to plastics, which can quickly deteriorate and weaken when exposed to the weather.

Having the station fully mounted, the software must be installed and configured in order for the EOS to begin monitoring climate. We can either perform all the steps manually or install a cloned image containing the operating system and all the software components required. In case of performing the process from scratch, first, the computing platform must have installed a compatible Linux distribution. Then, a web-server must be configured (e.g, Apache, Nginx), as well as, all dependent modules (PHP 7 and SQLite) installed in order to run the configuration application (the app just needs to be copied to a web directory). Also, some additional networking configuration may be required depending on the infrastructure deployment (e.g., setting a static IP address, firewall configuration, setting Virtual Hosts, etc.). The capturing software, which consists of a folder with several scripts, must be placed on a specific location (linked to the "Kit Config" app). Python 3 must be installed for I^2C and One-Wire communication. If using USB devices, other programming languages may be necessary, depending on the device's drivers and capturing script. For each connected device, its drivers must be installed and the designated capturing script must be placed inside the capturing software folder. The script's file name must match the model of the component. The station's initial database (an SQLite file) can be easily populated from a synchronization request to the server. If not possible, the database file can also be manually replicated from a similar EOS (i.e., using the same devices).

Connecting a new component to a working station requires the following steps:

1. Add information related to the device in the server. That information can be inserted

through the “Kit Insert” application;

2. Request a database synchronization from the EOS’s “Kit Config” application;
3. Properly connect the device to the EOS’s embedded system (soldering may be required, depending on the type of connection);
4. Install the device’s drivers and add its capturing script to the capturing software folder (the name of script must match the name of the device in the database). Some software tweaks may be needed in the capturing system, depending on the type of script.

After performing the described procedure, the new device should be detected by the system, and thus displayed in the “Kit Config” application. The EOS user can then edit its default configuration accordingly and start using the device. In case of not wanting to use a device, the user can deactivate it through the application and the capturing software will not execute its script.

Setting and deploying an Environmental Observing Station follows the presented strategy. However, it is a process which may not be so linear to achieve. There are complications that may arise, which require expertise and experience to be overcome. That is why it is expected that whoever performs it has minimal proficiency in Information Technologies.

Chapter 5

Implementation

In this chapter, we describe the implementation of a proof-of-concept EOS.

The initial goal was to perform environmental observing for two different scenarios: astronomy and meteorology. We assembled a prototype station integrating components related with both schemes. The physical structure of this hybrid EOS was built from scratch using off-the-shelf components. It integrates several meteorological components, as well as, an astronomy camera. Our prototype runs our capturing software framework and a web application for the configuration of its components. We also have a centralized server which is responsible for several tasks, such as, retrieving data from deployed EOS and providing visual content for each.

Our prototype is currently deployed at Parque Biológico de Gaia and has been configured to upload temperature, humidity and atmospheric pressure data to the Weather Underground service [1].

Both our server and the prototype have Internet access (Figure 5.1). Thus, users can access the applications from the Internet. The server and the EOS have communication between them to perform operations, such as, synchronization of databases and retrieval of collected data.

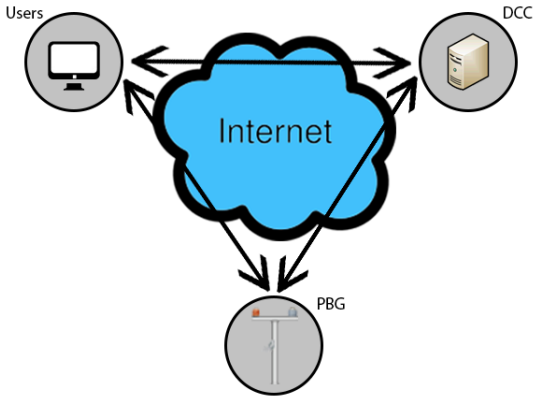


Figure 5.1: Communication diagram between users, the EOS and the server.

[1] https://www.wunderground.com/personal-weather-station/dashboard?ID=ISANTAMA1762

Regarding technologies used in this project, the web applications were developed in PHP with the Laravel framework, applying the MVC design pattern. We also implemented RESTful APIs to provide communication with the back-end services (through JSON). Thus, these services are not limited to the created applications and can be integrated in other systems. The Bootstrap library [14] was used to set the front-end layouts. Python and Linux shell scripting were also used for implementing server features (e.g., detection of plugged components). The software capturing framework was developed in Python and C++ and the Android application for the configuration of a station was built in Java using Android Studio.

5.1 Server

The centralized server is a Linux CentOS virtual machine hosted at the DCC data center with the following specifications: dual-core CPU, 1GB of RAM and about 35GB of storage capacity. It runs a web server available at http://dim.dcc.fc.up.pt with two web applications: one for visualization of captured data from remote stations and one for managing content in the database. The server is remotely accessible through SSH. Hence, we are able to manage and configure the database and retrieved data and check connectivity with each available station.

5.1.1 Database

The server has a centralized MySQL database configured to be accessed locally and from remote stations. This back-end database is used by the local web applications and it serves as an initial snapshot for each EOS's local database. It is also used for the synchronization of data with all stations.

The database has a hybrid schema: it adopts a relational data model, although adding flexibility with a few attributes of JSON type. The JSON format allows to associate different parameters to records of the same type (same table) without the need to create different columns or tables.

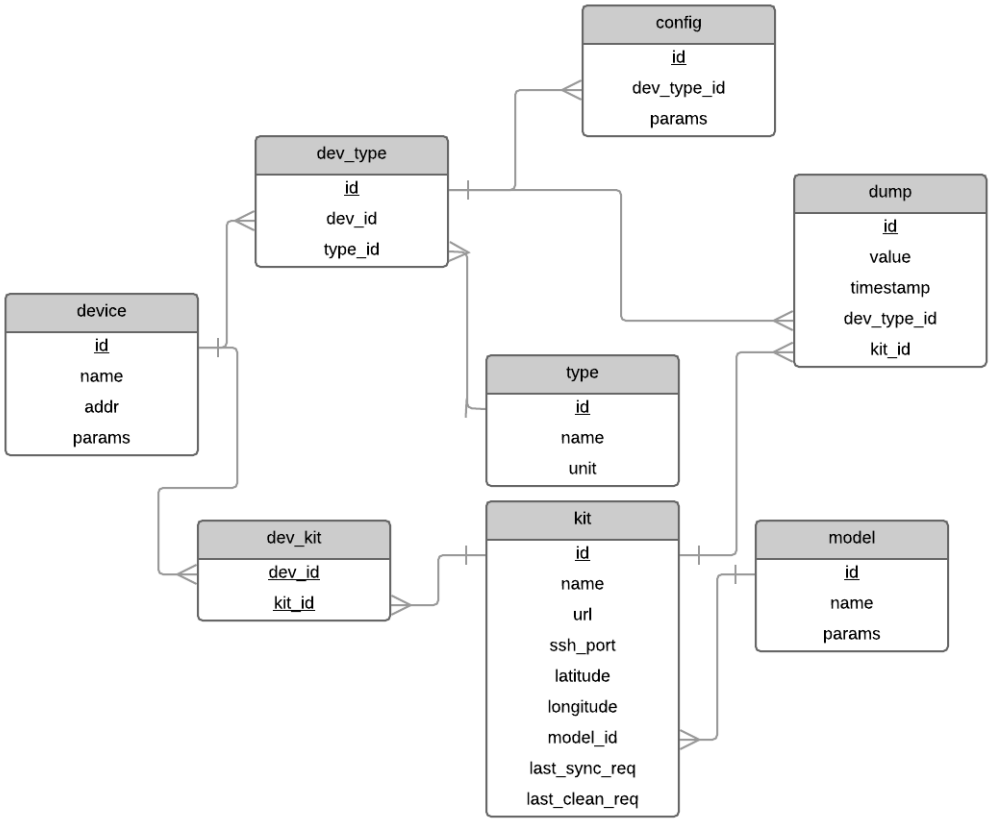


Figure 5.2: Database Relational Diagram.

The table 'device' stores information related to each device. For instance, a device can be a temperature sensor or a digital camera. The 'addr' column saves the device's corresponding address, in case it uses one. The 'params' field is of JSON type and it saves all the fixed parameters of the component. These parameters are not editable by users and have several purposes. This field can be used to include specific properties of each particular device. Furthermore, it can contain special JSON objects necessary for processing input. An object 'required' has the required fields to be edited by users and be part of a configuration, as well as, the input type associated to each field. A JSON object 'restrictions' includes limit values for the required fields. These required and restrictions objects are necessary to generate correct input fields on front-end interfaces and perform back-end input validation. An 'extra' object can be used to add extra information related to the device. Listing 1 is an example of a 'params' field for the camera ZWO ASI120MM.

```
{
"required":{
  "activate":"bit","delay":"int","gain":"int","exposure":"int",
  "bin":"predefined","dark":"bit","timelapse":"bit"},
"extra":{
  "dark_path":"/home/pi/all-sky/dark.png"},
"restrictions":{
  "activate":[0,1],"gain":{"min":0,"max":150},
  "exposure":{"min":0},"bin":["1x1","2x2","4x4"],
  "dark":[0,1],"delay":{"min":1},"timelapse":[0,1]}
}
```

Listing 1: Example of JSON fixed parameters (device ZWO ASI120MM).

The table ‘type’ stores all the data types of the information which can be collected from every component. Temperature, luminosity and pressure are a few examples. The table ‘dev_type’ is an auxiliary table that normalizes many-to-many relationships between devices and types, as a device can be associated with multiple types and vice-versa. For instance, our deployed station integrates a BME280 device which combines temperature, humidity and pressure sensors.

The table ‘config’ saves usage settings for each device/type relationship. The centralized database contains the default parameters which are initially set in each station’s local database. Posterior configuration changes are stored individually in each station’s local database.

The table ‘kit’ keeps information related to each deployed station, like it’s geolocation and accessible URL. Since a station/kit is a group of devices and the same device can be linked to multiple stations (devices of same model), the table ‘dev_kit’ handles the N:M (many-to-many) relation between the ‘device’ and ‘kit’ tables.

The table ‘model’ serves to group stations which have connected the same devices with the same properties. Each model can have a special specification defined in the JSON attribute ‘params’.

The table ‘dump’ stores the dump data recovered from the stations, after being parsed and validated. That information is then used by the visualization app. Each dump record has a value and a timestamp and associates the corresponding device/type and the EOS from which it was retrieved.

5.1.2 Content Visualization

Each deployed station performs environmental monitoring. All that captured information is retrieved by the server to be processed and made available for viewing. The “Kit View” web application is responsible for that synchronization of data between the server and the remote stations and for making that information visible to users.

The visualization interface organizes the information by station and, for each station, groups its devices by the corresponding data types. In case it is a camera, it displays the most recent captured image or a generated timelapse video, depending on the settings. For the other types of data, charts are created to give a timeline perspective for each device used. We implemented the charts using Javascript and Chart JS library [15]. The web app is dynamic, as all the displayed content is retrieved from the server's database.

Figure 5.3 shows a time series chart for the temperature devices connected to our deployed station (named "PBG"). As illustrated, we have a BME280 with the address 0x77 (I^2C interface) and a DS18B20 with the address 28-0416c1ee29ff (One-Wire interface).

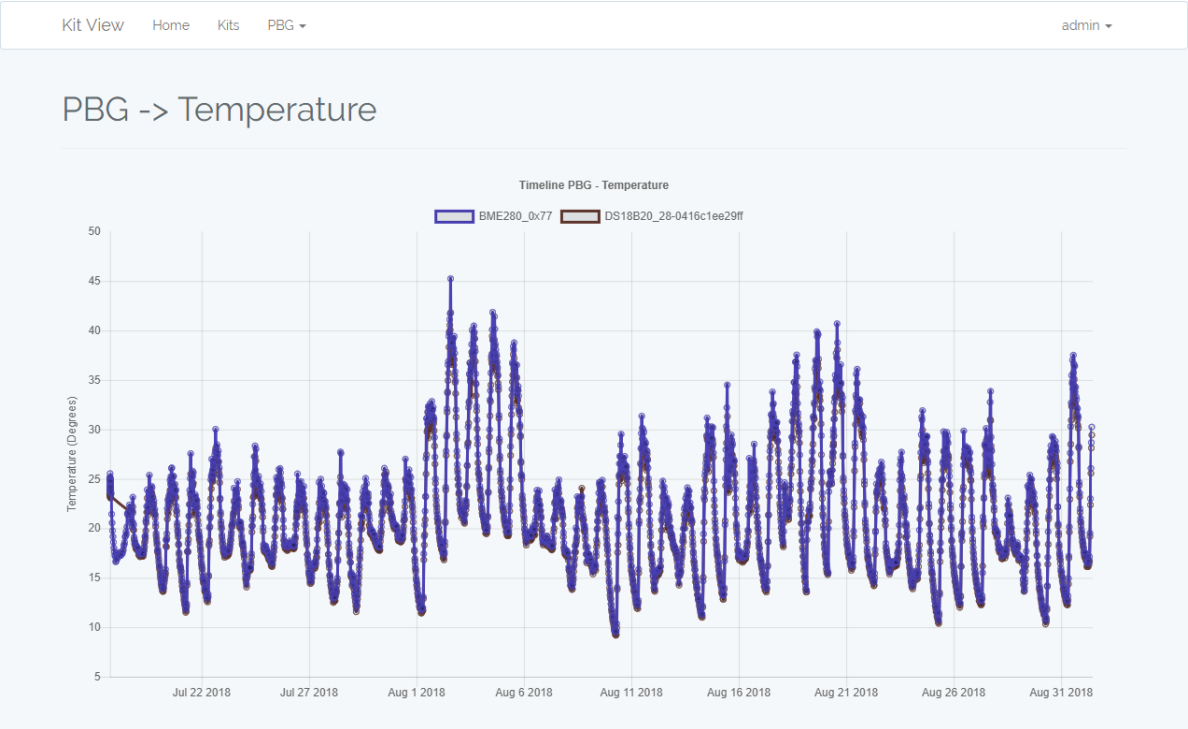


Figure 5.3: "Kit View" PBG - Temperature chart.

Figure 5.4 refers to the Camera tab, displaying the latest picture taken from our camera, ZWO ASI120MM.

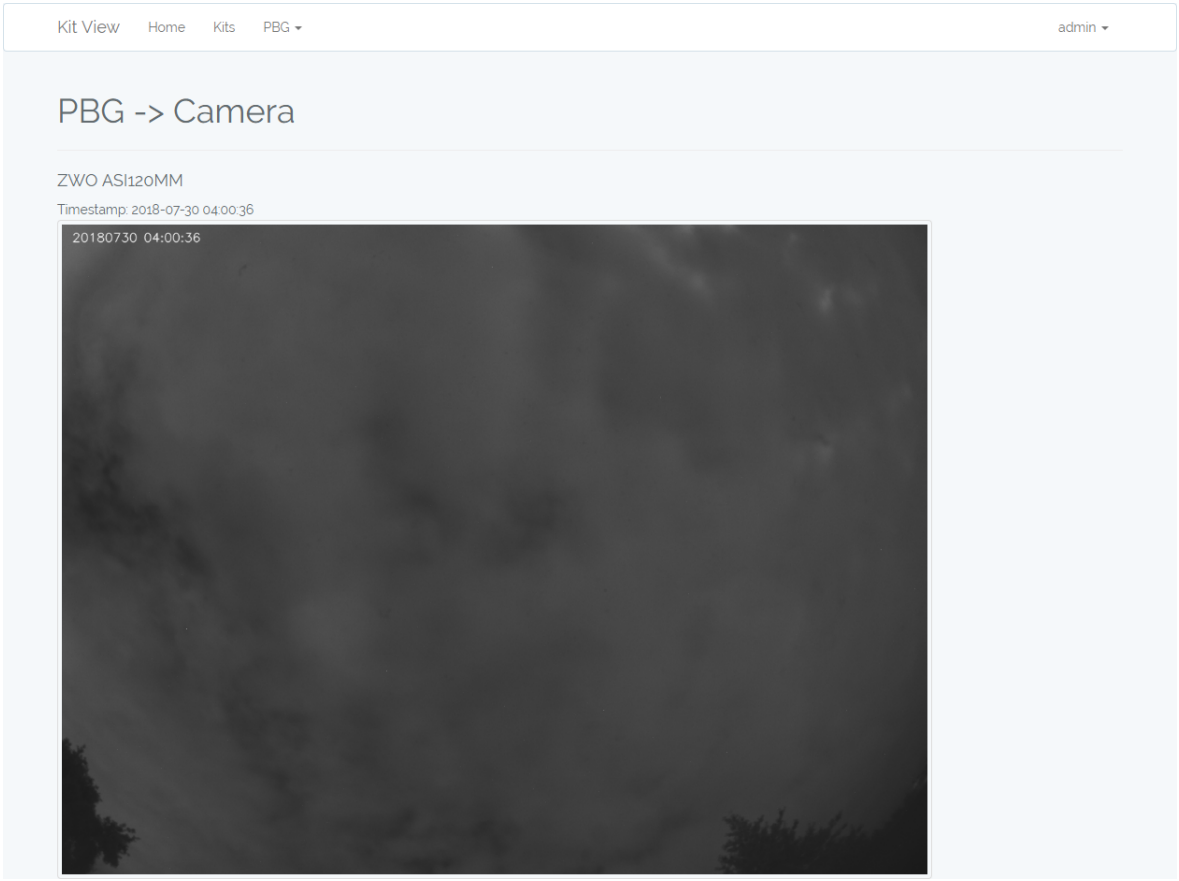


Figure 5.4: "Kit View" PBG - Camera's latest picture.

"Kit View" also features dump synchronization and data cleansing operations per station (Figure 5.5). An authenticated user has the ability to request synchronization with a specific EOS, in order for the server to store/display updated content. The user can also request the server to perform data cleansing for that specific station, deleting dump information older than the specified months (default is 6 months).

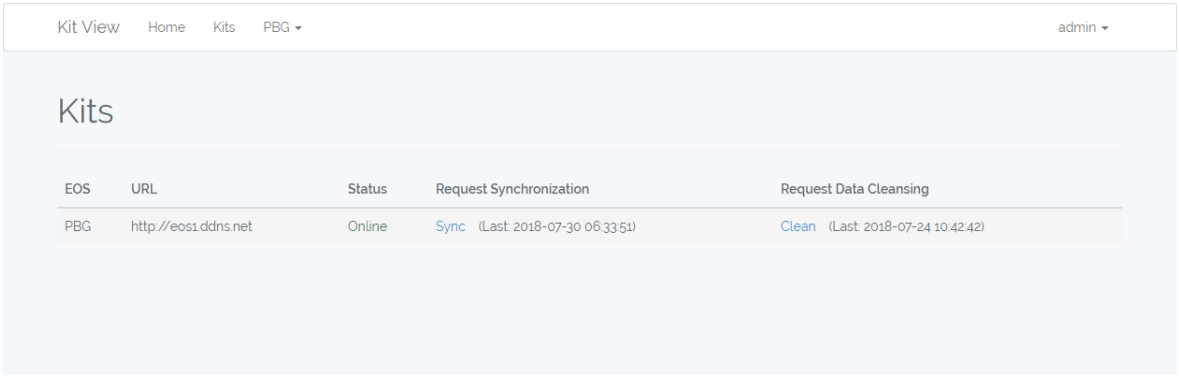


Figure 5.5: "Kit View" - All deployed EOSs overview

The retrieval of data from each station can be done in two ways: through HTTP (or HTTPS)

GET requests or by SSH. The former requires for the EOS dump files to be public. The server performs incremental downloads of those files (using wget utility) followed by data processing and database insertion. The latter creates a SSH session for each request, using public key authentication to retrieve the data (using rsync utility). For security reasons, by default, we disable password authentications and only allow public-key authentication method. Performing automated SSH sessions introduces some login security issues, since it requires non-privilege users (web-server user in our case) to have unwanted permissions. For that reason, we have implemented the two alternatives. As we require public access to the raw data dumps generated by our prototype and have it accessible through HTTP, the synchronization is currently done by the first method described.

Listing 2 shows a synchronization request example. In this case, is for our PBG station. The "1" refers to the ID of the EOS in the database. Therefore, each GET /kits/{id}/sync request is only for the station specified by the ID flag.

```
1 GET http://dim.dcc.fc.up.pt/kit_view/kits/1/sync
```

Listing 2: "Kit View" Sync request example (HTTP GET).

The data cleansing operation can be requested on-demand. Nonetheless, the system is set to periodically perform data cleansing in order to prevent the database from reaching exponential sizes and causing losses of performance. The GET request takes two arguments: the ID of the EOS and the number of months old from which all the older dump records are to be deleted. The structure of the GET request is as follows: GET /kits/{id}/clean/{n}. Listing 3 shows an example of a data cleansing request for the EOS with ID "1" to delete dump records older than 6 months.

```
1 GET http://dim.dcc.fc.up.pt/kit_view/kits/1/clean/6
```

Listing 3: "Kit View" Data Cleansing request example (HTTP GET).

Both sync and data cleansing operations require an authenticated user and are protected against cross-origin attacks (through the use of CSRF tokens). If attempted, the user is redirected to the login page.

5.1.3 Content Management

When setting up a new station, its local database is derived from the centralized database. It clones all the information related to the accepted devices and data types, as well as, default usage configurations for each device/type relation. Additionally, whenever a new device needs to be

attached to a specific station, it needs to be previously added in server's database. In a typical scenario, whoever will be using the observing station may not be responsible for setting up and making the station operational. We, or someone with the necessary knowledge, prepare and set up the station and, afterwards, the configuration of components can be easily made through the "Kit Config" application as intended. Managing content of the database can be achieved with direct access to the server and the usage of scripts and/or the Database Management System (DBMS). However, it is not a method intended for the average user. Hence, with the goal of simplifying the process of adding/editing information and offering the ability to easily view the database's content, we developed a web application "Kit Insert". This app offers a control panel from where a user can easily view content from the database and, if authenticated, can also edit or add new values. "Kit Insertion" serves as our own Content Management System (CMS), having the peculiarity of not supporting the removal of records. For that purpose, direct access to the server is required. Hence, an EOS user can safely use the application with no risk of deleting data.

The application also provides a RESTful API for getting content in JSON format. For example, listing 4 shows a GET request for retrieving information about all devices recognized by the system. Listing 5 refers to a GET request for a specific device (ID of value 2) and Listing 6 illustrates the server's response.

```
1 GET http://dim.dcc.fc.up.pt/kit_insert/api/devices
```

Listing 4: "Kit Insert" - Example of an API request (for all devices).

```
1 GET http://dim.dcc.fc.up.pt/kit_insert/api/devices/2
```

Listing 5: "Kit Insert" - Example of an API request (for device of ID=2).

```
{
    "id":2,
    "addr":"0x77",
    "name":"BME280",
    "params":"{
        \"required\":{
            \"activate\":\"bit\",
            \"delay\":\"int\"
        },
        \"restrictions\":{
            \"activate\":[0,1],
            \"delay\":{\"min\":1}
        }
    }",
    "created_at":"2018-07-20 15:35:14",
    "updated_at":"2018-07-20 15:35:14"
}
```

Listing 6: “Kit Insert” - Example of a server API response (for device of ID=2).

Table 5.1 lists all the available API requests in “Kit Insert” application.

Method	Route	Description
GET	/api/configs	Get all default configurations
GET	/api/configs/{id}	Get specified default configuration (by ID)
GET	/api/configs/{id}/dev_type	Get the device/data type relation of the specified configuration
GET	/api/configs/{id}/device	Get the device of the specified configuration
GET	/api/configs/{id}/type	Get the data type of the specified configuration
GET	/api/dev_kits	Get all device/kit relationships
GET	/api/dev_types	Get all device/data type relationships
GET	/api/devices	Get all devices
GET	/api/devices/{id}	Get specified device (by ID)
GET	/api/devices/{id}/configs	Get all the configurations of the specified device
GET	/api/devices/{id}/dev_kits	Get all device/kit relations of the specified device
GET	/api/devices/{id}/dev_types	Get all device/data type relations of the specified device
GET	/api/dumps	Get all dump records
GET	/api/kits	Get all EOS
GET	/api/kits/{id}	Get specified EOS (by ID)
GET	/api/kits/{id}/dev_kits	Get all the device/kit relations of the specified EOS
GET	/api/kits/{id}/dumps	Get all dumps of the specified EOS
GET	/api/kits/{id}/model	Get the model of the specified EOS
GET	/api/models	Get all models
GET	/api/models/{id}	Get specified model (by ID)
GET	/api/models/{id}/kits	Get all kits of the specified model
GET	/api/types	Get all data types
GET	/api/types/{id}	Get specified data type (by ID)
GET	/api/types/{id}/configs	Get all configurations of the specified data type
GET	/api/types/{id}/dev_types	Get all device/data type relations of the specified data type

Table 5.1: Table of “Kit Insert” list of API’s request.

In terms of layout, we developed a simple and uniform interface. It is organized in modules displayed in the form of tabs and for each, the content visualization is shown in a table (Figure 5.6) and the insertion/edition of records is done through input forms (Figure 5.7).

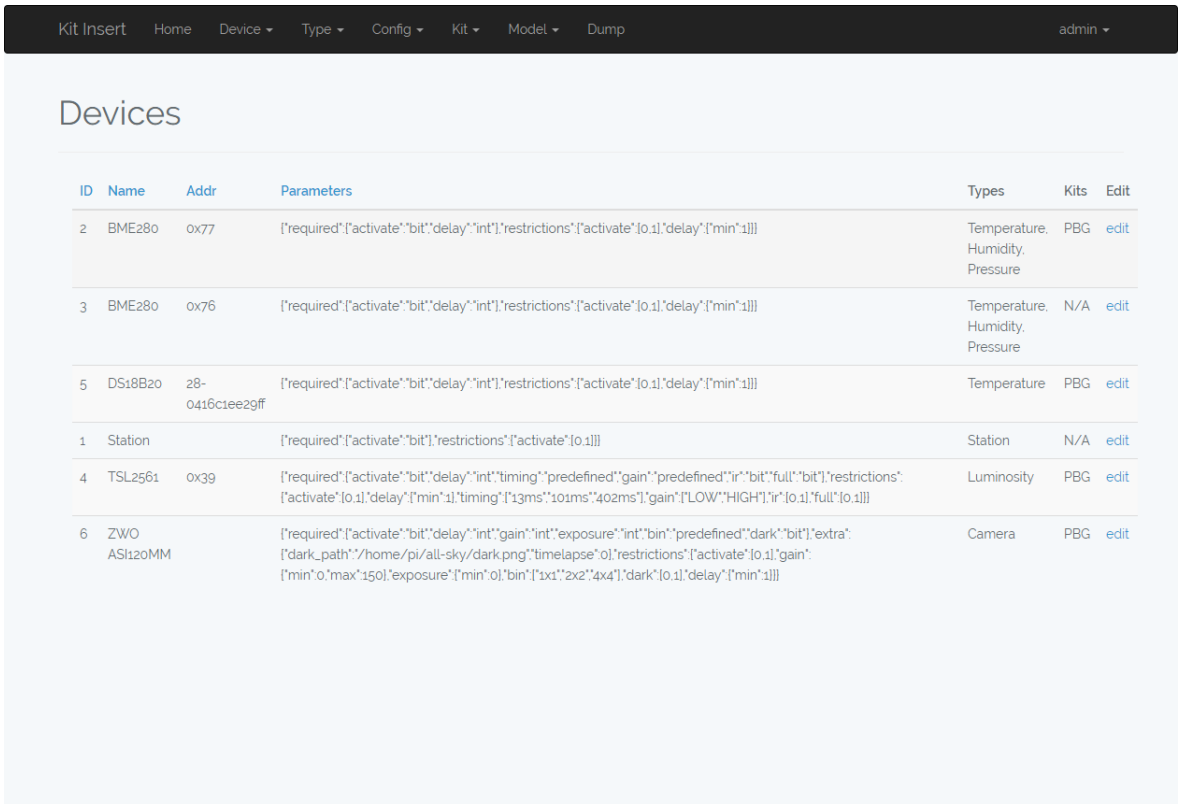


Figure 5.6: “Kit Insert” - Table of Default Configurations.

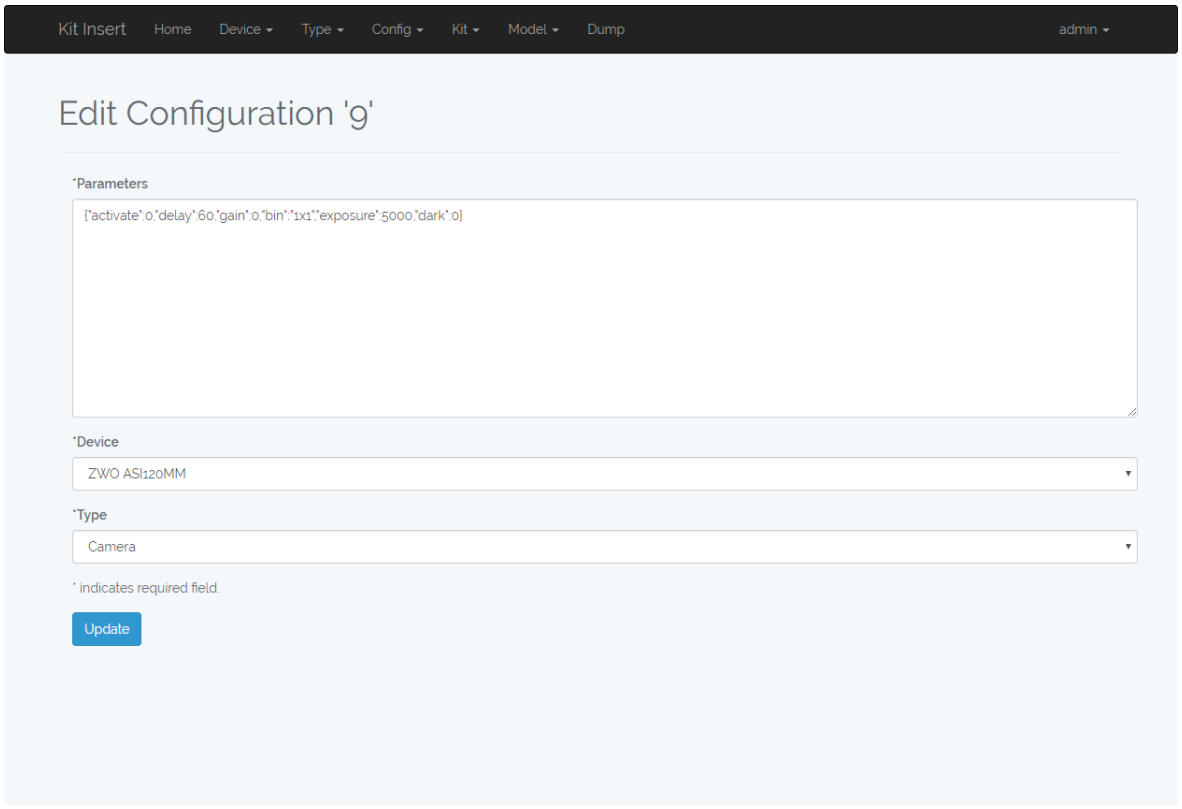


Figure 5.7: “Kit Insert” - Edition form of a configuration.

All the user input is passed through a validation process before being submitted to the database. The client interface already limits required fields and correct data types within a specific range (based on the parameters set for each device). After the server's validation, the client receives a feedback response on whether the insertion/update was successful or if it failed, reporting the error.

The reason we opted for a centralized management was for reducing complexity. If every station had an independent database, adding a new device model on several EOSs or update some common parameter on a device present in multiple EOSs would require to manually do it on each one. With a centralized database, we only need to perform it once. Furthermore, it eases the ability to automatically propagate the new changes to each online station. Each station has also the ability to force on-demand synchronization with the server.

In order to automate the process of building the database, we have developed scripts for creating and setting up the tables (known as migrations). If for some reason, we require to reset the database, we can simply delete it and rerun the scripts. Also, an interesting feature of the Laravel framework is that it allows to quickly rollback applied migrations. It is particularly useful when changes produce application errors and a single command can revert the database to a previous state.

Alongside the migrations, we have also created scripts to populate the database with the necessary information (known as seeders). At the moment, it includes the devices which we use in our prototype, their default configurations and information about the EOS itself. Thus, it translates into all the content which is currently displayed on the "Kit Insert" app. In the future, as we add more devices/stations, we can update the seeder files in order to auto-set the database with the default content, in case it needs to be rebuilt.

5.2 Station

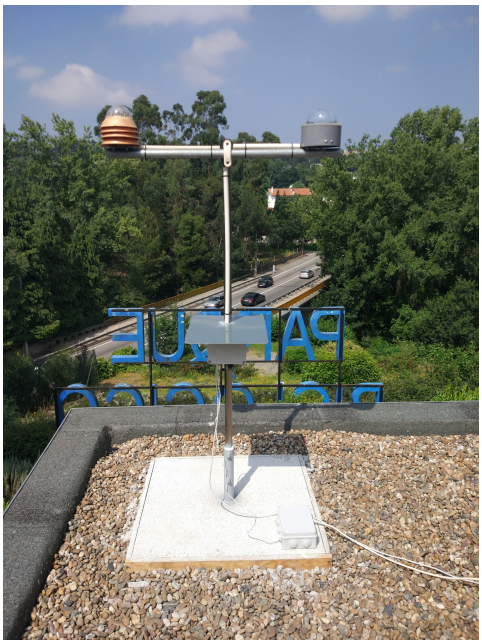
The station at Parque Biológico de Gaia is a handmade structure using off-the-shelf materials and integrates some components acquired for this specific project. The intention was to create a low-cost hybrid scenario observing station (for astronomy and meteorology) while operating our designed framework. The devices used are a ZWO ASI120MM astronomy camera [8], a DS18B20 temperature sensor (One-Wire interface) [27], a BME280 I^2C/SPI sensor (for temperature, humidity and pressure) [12] and the luminosity sensor TSL2561 (I^2C interface) [75]. As previously mentioned, a Raspberry Pi 3 Model B is used as the main computing platform.

The station is installed on top of the roof of the astronomical observatory (Figure 5.9). It is powered by an AC adapter and is connected to the Internet through an Ethernet cable. Its configuration server is online and can be accessed at http://eos1.ddns.net. Figure 5.8 illustrates our fully functional prototype during the experimentation stage (left) and at Parque Biológico de Gaia (right). The testing/troubleshooting period lasted a month, from 24th of April until 25th of May. The EOS was deployed at Parque Biológico de Gaia in 17th of July and has been

operational ever since.

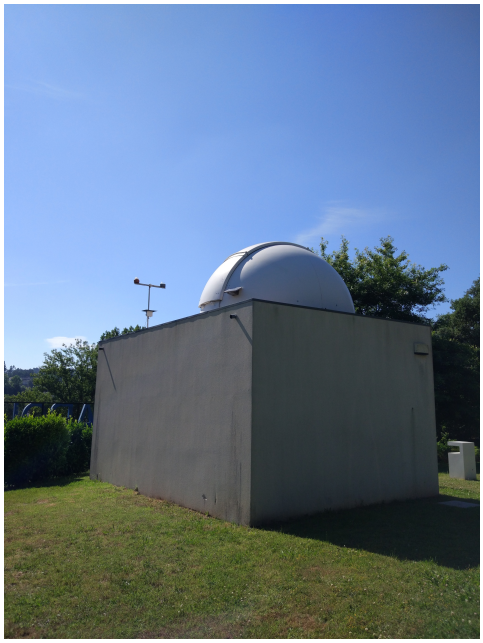


(a) Station deployed during experimentation stage.



(b) Station currently deployed at Parque Biológico de Gaia.

Figure 5.8: Station Deployment.



(a)



(b)

Figure 5.9: Astronomical Observatory at Parque Biológico de Gaia.

5.2.1 Physical Structure

The physical structure was built using off-the-shelf materials. It has a shape of a "T". The main mast column is a stainless steel pipe with about 2m of length. On top of it stands a horizontal tube with approximately 1m of length. This horizontal tube supports the connected devices on its extremities.

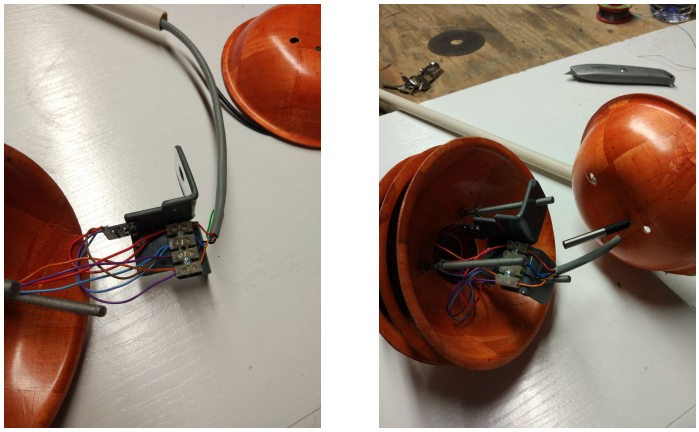
On weather stations, temperature, humidity and pressure sensors are usually placed inside a solar radiation shield (or a Stevenson screen). There, the devices are protected from precipitation and direct exposure with the sun. The shield's shape allows enough air flow to pass through the sensors in order to make accurate reading of variables. As commercial solar radiation shields are frequently overpriced and with the goal of enriching our project, we designed our own radiation shield. We bought 4 plastic bowls and stacked them with 3 threaded bars, as illustrated in Figure 5.10. The usage of nuts allows to set flexible heights between each bowl so the items inside can fit.



(a) (b) (c)

Figure 5.10: Solar radiation shield assembly.

Both TSL2561 (luminosity) and BME280 (temperature, humidity and pressure) sensors communicate through the same digital I^2C protocol and can share the same channels. So, instead of passing 2 separate cables, we opted for a single cable. Inside the shield, we placed a small strip of electrical terminal connectors (Figure 5.11). The connectors join both devices to the 4 wires of the cable. We had to solder 4 wires to the 4 corresponding I^2C contacts in each component. The purchased DS18B20 sensor was the the waterproof version, so it already included a PVC cable soldered to the device. However, the cable was too short (1m) and we had to solder a small extension cable to run inside the tube.



(a) (b)

Figure 5.11: Connection of sensors inside the solar radiation shield.

Unlike the other sensors, the luminosity sensor requires exposure. We needed to engender a way to stand it on top of the upper bowl. So, we installed a small transparent plastic dome with the TSL2561 standing inside. We applied silicone around the borders to secure the dome above the bowl and prevent water from entering. Our solar radiation shield ended up as illustrated in Figure 5.12.



Figure 5.12: Homemade solar radiation shield.

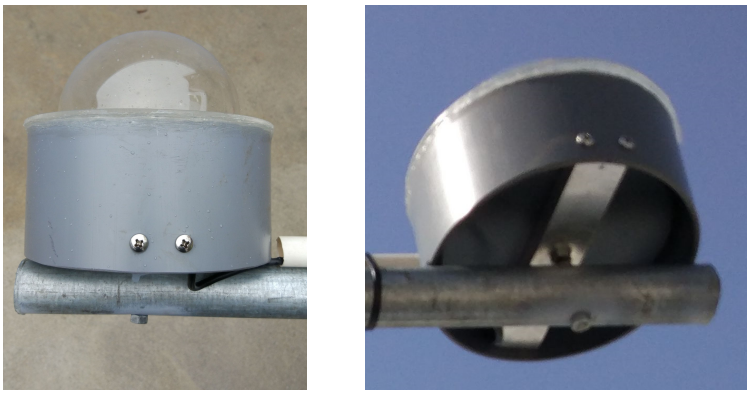
We also had to come up with a platform to support the astronomy camera. We acquired a transparent acrylic dome with 16mm of diameter. We then used a small segment of a 16mm PVC pipe to secure the dome on the edge of the tube. We cut a circle with the same diameter from a PVC panel and carved a ring in the center with the same diameter of the camera lens. This circular piece was fixed to the pipe and to the dome with the help of silicone. As illustrated in Figure 5.13, taking advantage of the camera's mounting slot, we created a mounting bracket using a threaded bar. We made sure it had more than the necessary height. In this manner, it is not limited to this specific camera and any other camera with the same mounting slot can be attached.



(a) (b) (c)

Figure 5.13: ZWO ASI120MM support platform assembly.

The platform holding the camera is fixed to the PVC pipe using 4 screws, 2 on each side. The threaded bar is also used to install the whole gear on the horizontal tube in the main structure (Figure 5.14).



(a) (b)

Figure 5.14: Camera support platform.

For the main board, we bought an electric junction box. Inside, we set up a plain PVC base stand and screwed the Raspberry Pi to it, as well as, a strip of electrical terminals where we connected the cable wires and the pins. We also placed a PVC flat plate directly above the box, serving as a shield for solar radiation. The purpose was to try to keep the box and the components inside as cool as possible. In addiction to the insulation rubber around the cover, we also planted a moisture absorber packet inside the case.

In order to facilitate the access to the control panel, we placed the box at about 1.7m height (assuming the base of the main tube as the start). We arranged the components and the case having in consideration the possibility of adding more devices or pairing a powerbank in the future (Figure 5.15).



(a) (b)

Figure 5.15: Control panel.

We connected the powering pair wires (3.3V and ground) of both cables to the terminals where the corresponding Raspberry Pi pins were linked. We attached the I^2C data lines (SDK and SDA) of the pins of the Raspberry Pi and the 2 related wires of the I^2C cable over the associated connectors. For the One-Wire interface, we connected a pull-up resistor of 4.7 ohm between the 3.3V supply and the One-Wire data channel.

5.2.2 Station Preparation

Creating and deploying an EOS involves a certain number of steps. A station needs a Raspberry Pi (or a similar SBC) configured with at least the essential software. That software includes a SQLite database, a web application for the configuration of its devices, the capturing software framework and the required drivers for the attached components. Obviously, the software may have some dependencies, which need to be also installed. The digital interfaces that are to be used (I^2C and One-Wire, for instance) need to be software enabled before connecting the components.

We have developed scripts to automate and lighten the configuration process. For the components used in this project, we have scripts for installing the needed device drivers, as well as, the SDK used by the capturing framework. Additionally, for each device, we have the related script to be integrated in the capturing framework. Regarding the local database, we created scripts for generating the local database (migration files). Filling the database is done through the synchronization process: a connection to the server is made and information is transferred and inserted into the local database. Since that local database is represented by a file (a SQLite database), it is also easy to be propagated to other stations, which have the same components. The user only has to request the custom kit (specify which components are to be used) and, after the station is all set, configure the usage of each component through the application.

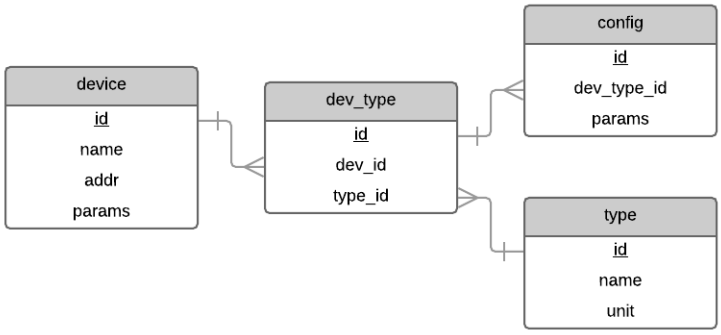


Figure 5.16: Local Database Relational Diagram.

Each station only has a view of all the known devices, data types and the corresponding configurations (Figure 5.16). It does not have or need a perspective of the other deployed EOS.

5.2.3 Station Configuration

Each station runs a web application "Kit Configuration". Its purpose is to offer users an interface for customization of configuration parameters. Each recognized device is displayed, listing its usage configuration parameters. Figure 5.17 illustrates a configuration form for the TSL2561 device. Authenticated users have the ability to activate/deactivate the whole station, individual components and alter specific properties for each. There is also the possibility of requesting a database synchronization with the central server through a click of a button (Figure 5.18). Due to the centralized server being behind a firewall and not allowing connections from outside the DCC domain, we cannot connect to the remote MySQL server directly. The only way to connect to the MySQL server is by a SSH tunnel with local port forwarding. So, the database synchronization process is done through the RESTful API provided by "Kit Insert" application. It consists of an automatic mechanism which retrieves data related to all devices, data types and default configurations. It checks whether a new device/data type is not stored in a station's database and inserts it, as well as, its corresponding relationships and default configurations. If any component in the station's database is no longer present in the server, its data gets deleted. Thus, the local database is able to be synchronized with the remote one.

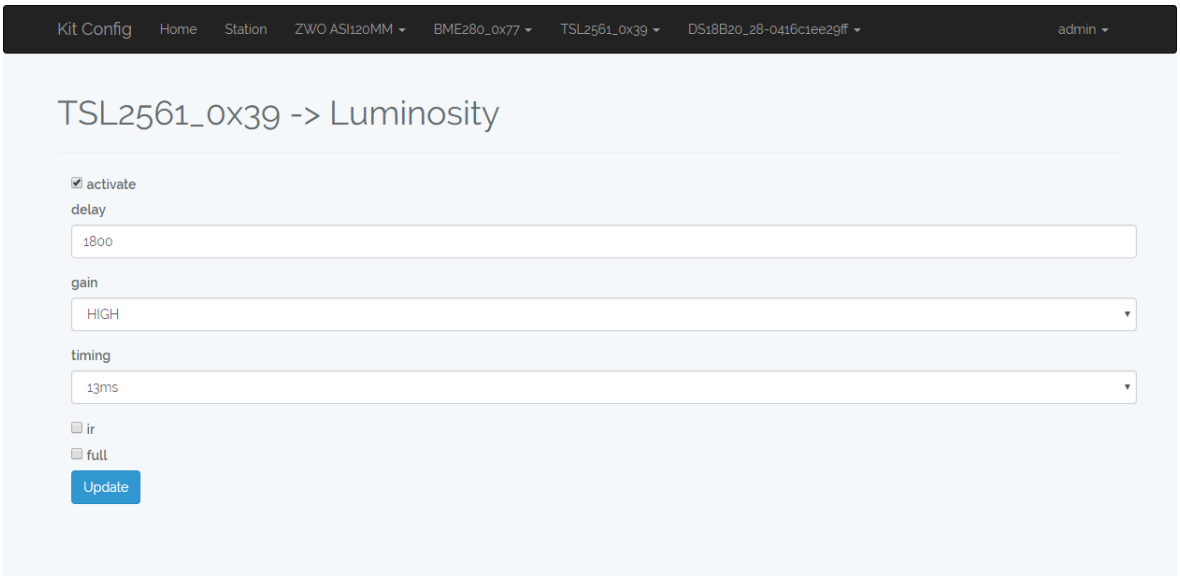


Figure 5.17: "Kit Configuration" - Configuration Interface (TSL2561 device).

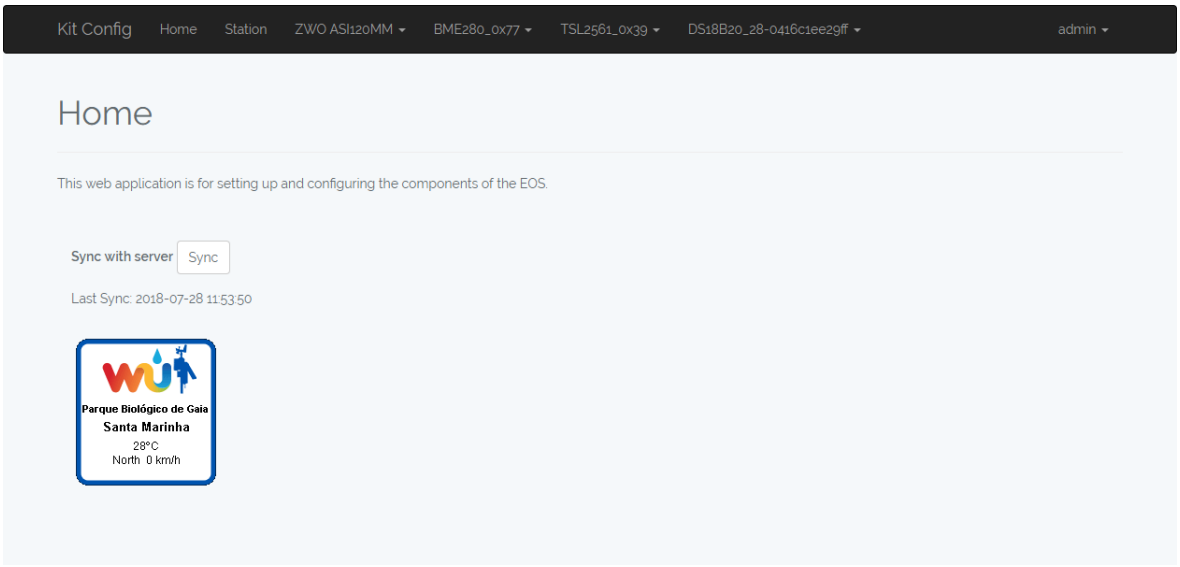


Figure 5.18: "Kit Configuration" - Home Page (Sync Request and WUnderground access).

Whenever a client loads the app, the server performs a recognition of plugged devices. We developed a Python script to detect connected I^2C and One-Wire devices (Listing 7). For the I^2C interface, we basically send a broadcast message on the bus for every address within the address range (in this case, 0-127 for 7 bit addresses). Each received response contains the address of the sender. Regarding the One-Wire protocol, all hooked up devices are represented as a file in "/sys/bus/w1/devices/" directory. The name of the files correspond to the device's address.

```python
#!/usr/bin/python3
import smbus
import glob

def getPluggedI2CDevices(devs=[],only_print=False):
    bus = smbus.SMBus(1)

    for dev in range(128):
        try:
            bus.read_byte(dev)
            addr = hex(dev)
            if only_print:
                print(addr)
            else:
                devs.append(addr)
        except:
            pass
    return devs

def getPlugged1WireDevices(devs=[],only_print=False):
    search_dir = "/sys/bus/w1/devices/"
    one_wire_devs = glob.glob(search_dir+"28-*") #addresses start with 28
    for dev in one_wire_devs:
        pos = dev.find("28-")
        dev = dev[pos:]
        if only_print:
            print(dev)
        else:
            devs.append(dev)
    return devs

def main(args):

    if args[0]=="I2C":
        getPluggedI2CDevices(only_print=True)
    if args[0]=="1W":
        getPlugged1WireDevices(only_print=True)
if __name__ == "__main__":
    import sys
    main(sys.argv[1:])
```

Listing 7: Python script for detecting I^2C and One-Wire devices.

For the detection of USB cameras, we implemented two possibilities. We developed a C script to detect USB plugged cameras from ZWO manufacturer (Listing 8). It uses the company's SDK and a C API to retrieve the model of the plugged cameras. The problem with this method is that it only works for devices from ZWO. The other alternative is scanning the USB ports (with the command "lsusb -v") and parsing specific fields like "bDeviceClass" or "iProduct". This is a more generic approach, but depending on the manufacturer, device and the drivers installed, devices of the same category can have different fields set up, making them more difficult to be identified.

```c
#include<stdio.h>
#include "ASICamera2.h"

int  main(int argc, char* argv[])
{
    int numDevices = ASIGetNumOfConnectedCameras();

    ASI_CAMERA_INFO ASICameraInfo;

    for(int i = 0; i < numDevices; i++)
    {
        ASIGetCameraProperty(&ASICameraInfo, i);
        printf("%s\n",ASICameraInfo.Name);
    }

    return 0;
}
```

Listing 8: C script for detecting ZWO cameras.

Each recognized device is mapped to its related record in the database. From its address, we are able to get the name of device, properties, configuration parameters and associated data types. A successful mapping makes the device detected and, therefore, ready to be configured in the "Kit Configuration" application.

We have also implemented an API in the "Kit Configuration" app. Its purpose is to allow communication between the Android app and the back-end server. Table 5.2 details the RESTful API in the "Kit Configuration" app.

Method	**Route**	**Description**
GET	/api/configs	Get all configurations
GET	/api/configs/{id}	Get specified configuration (by ID)
POST	/api/configs/{id}	Submit and update a specified configuration (by ID)
GET	/api/dev_ack	Get all devices which are currently detected
GET	/api/dev_discover	Get all devices/addresses which are currently plugged and recognized

Table 5.2: Table of "Kit Configuration" RESTful API.

Listing 9 illustrates a server response example to a request for Device Discovery (GET /api/dev_discover). Listing 10 shows a server response example to a request for Device Detection (GET /api/dev_ack).

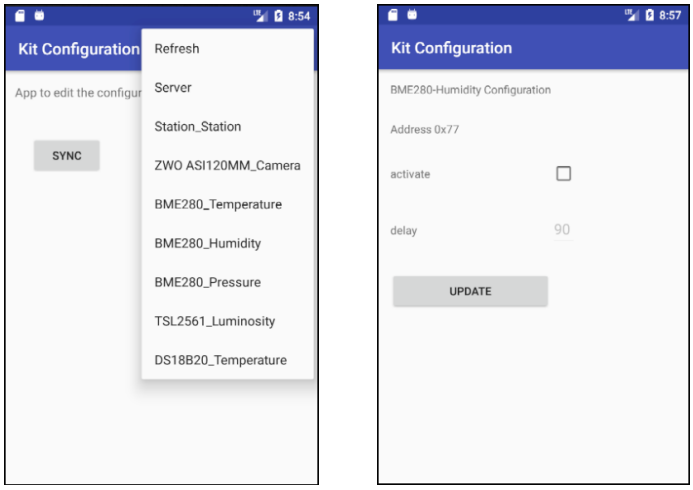
```
{
    "cameras":["ZWO ASI120MM"],
    "i2c":["0x77","0x39"],
    "one_wire":["28-0416c1ee29ff"]
}
```

Listing 9: “Kit Configuration” - Example of a server response (GET /api/dev_discover).

```
{
    "cameras":[
        {"id":6,
        "name":"ZWO ASI120MM",
        "addr":null,
        "params":"{
            \"required\":{...},
            \"extra\":{...},
            \"restrictions\":{...}}",
            "created_at":"2018-07-28 11:53:47",
            "updated_at":"2018-07-28 11:53:47"
        }
    ],"i2c":[
        {"id":2,
        "name":"BME280",
        "addr":"0x77",
        "params":"{
            \"required\":{...},
            \"restrictions\":{...}}",
            "created_at":"2018-07-28 11:53:46",
            "updated_at":"2018-07-28 11:53:46"
        },{"id":4,
        "name":"TSL2561",
        "addr":"0x39",
        "params":"{
            \"required\":{...},
            \"restrictions\":{...}}",
            "created_at":"2018-07-28 11:53:46",
            "updated_at":"2018-07-28 11:53:46"
        }
    ],"one_wire":[
        {"id":5,
        "name":"DS18B20",
        "addr":"28-0416c1ee29ff",
        "params":"{
            \"required\":{...},
            \"restrictions\":{...}}",
            "created_at":"2018-07-28 11:53:47",
            "updated_at":"2018-07-28 11:53:47"
        }
    ]
}
```

Listing 10: “Kit Configuration” - Example of a server response (GET /api/dev_ack).

Besides the web app, we have also developed a native Android application for the configuration of an EOS. It communicates with the "Kit Configuration" server using the designed API. The user has the ability to set the station's server URL and port and, if valid, the associated data is retrieved. Hence, the application can be used to configure all detected components in the station. A dynamic menu lists all devices ready to be used on the specified EOS (Figure 5.19). Each item redirects to a new activity where all the configuration parameters are loaded into a form. The user has the ability to change those properties and request an update with the server. The saving process uploads the new configuration to the server (using JSON). After all the input has been validated, the application gets feedback from the server and informs the user through a pop-up message.



(a) (b)

Figure 5.19: "Kit Configuration" Android app.

5.2.4 Capturing Data

We developed a software framework for capturing data from any component plugged to an EOS. For the addressable devices (using I^2C/One-Wire protocols), we implemented a platform using Python. Its execution follows a structured order:

1. Verify if station is online: check station's configuration in the local SQLite database; if online, proceeds; otherwise, halts;
2. Detect connected I^2C/One-Wire devices using the previously referred methods;
3. Mapping detected devices to their configuration: through each device's address, retrieve their related information from the local database; in case a device is not yet on the database, a message is displayed; the script continues as long as it has at least 1 detected device;
4. Start capturing data from each detected device: for each device, load its related script; for

each script, an execution thread is created; the capturing data is printed on the screen and written to a text file (Listing 11).

```
2018-05-07 16:39:52    74.558
2018-05-07 16:40:52    73.612
2018-05-07 16:45:26    72.974
2018-05-07 16:46:26    72.227
2018-05-07 16:47:26    72.346
2018-05-07 17:11:52    69.897
2018-05-07 17:12:52    69.410
2018-05-07 17:34:35    71.454
2018-05-07 17:35:35    71.552
2018-05-07 17:36:35    71.406
2018-05-07 20:30:30    99.607
2018-05-07 20:31:30    99.699
2018-05-08 10:54:32    85.006
2018-05-08 10:55:32    84.827
2018-05-10 15:40:29    55.442
2018-05-10 15:41:29    56.383
2018-05-10 15:42:29    55.550
2018-05-12 10:04:30    85.282
2018-05-12 10:05:30    85.644
2018-05-12 10:06:30    85.606
```

Listing 11: Humidity dump file example (device BME280).

We implemented a "DeviceConfig" class to assign the necessary parameters for each component. The capturing framework creates an instance for each detected device and loads it with the device's retrieved properties.

In order for each device to be used by the framework, a corresponding Python script (with the file's name matching the device's) must exist in the scripts directory. Each script receives as an argument an instance of "DeviceConfig" and communicates with the device's drivers through the designed Python API.

Our deployed station uses an astronomy camera ZWO ASI120MM. For capturing pictures, we developed a C++ script using ZWO SDK [9], based on available examples and Thomas Jacquin's project [37]. For processing images, we used OpenCV library [52] and for processing JSON format in C++, we used a JSON parser available at GitHub [38]. This script checks for plugged ZWO cameras, loads the corresponding model's configuration from the local database and then proceeds with capturing pictures at the given cadence. In case the dark subtraction option is enabled, the script checks if the dark image file already exists; if it exists, proceeds with the capture, subtracting the dark image on every frame taken. Otherwise, it requests to first take a dark picture. The script halts until an enter is pressed. The user, then, must cover the camera lens (with a dark cloth, for example) and press the enter key when ready to take the picture.

In order to automate the execution of the astronomy capturing scripts only during the

astronomical twilight [2], we created scheduled tasks using crontab and Sunwait [73]. Sunwait is a C program which calculates timings for sunrise, sunset, as well as, civil, nautical, and astronomical twilights for a given timezone and geolocation (latitude and longitude). There are two scheduled tasks: one for starting the script and the other for terminating it. Each day at 18:00, Sunwait program starts running. Its purpose is to wait until the start of astronomical twilight. Then, the capturing script is executed until the second scheduled task. The second crontab rule starts every day at 04:00 with Sunwait program, waiting for the end of the twilight. At that instant, a signal is sent to the capturing script for it to terminate.

```
0 18 * * * ${HOME}/sunwait/sunwait astr stop -1:00 41.097092N -8.55608W;
    ${HOME}/all-sky/capture
0 4 * * * ${HOME}/sunwait/sunwait astr start -1:00 41.097092N -8.55608W;
    kill $(ps aux | grep "capture" | awk '{print $2}')
```

Listing 12: Crontab scheduled tasks.

The (41.097092, −8.55608) coordinates correspond to the location of Parque Biológico de Gaia, where the station is deployed. The $-1:00$ sets an offset of $+1$ hour. It is required due to the calculations being made in UTC time zone and our time zone being WEST.

5.2.5 Weather Underground Integration

Since our prototype uses some meteorological devices, we determined it would be advantageous to create our own Weather Underground station [3] and contribute to a worldwide community. After we registered at the service, we set up a new personal station with the coordinates of the deployment location.

Each station is identified by an ID and a key. Both are required to send data to the server. The upload protocol uses a HTTP GET request [84]. As parameters, we send the authentication credentials (station ID and key), the timestamp and the weather condition types with the corresponding values. Currently, we upload data for humidity, pressure and for two temperature components. Additionally, we calculate the dew point from the temperature and humidity values. The formula used is presented below (from [21]). T, f and T_p refer to temperature, relative humidity and dew point values, respectively.

$$T_p = (\frac{f}{100})^{\frac{1}{8}}(112 + 0.9T) + 0.1T - 112$$

We developed a Python module to integrate in our capturing framework (Listing 13). This module is run on a separate thread and, periodically, checks the latest captured values from

[2] https://www.timeanddate.com/astronomy/astronomical-twilight.html
[3] https://www.wunderground.com/weatherstation/overview.asp

the corresponding device's dump files. From the specified devices to upload, if the most recent capture is within the latest 30 minutes, that value is selected. Therefore, a request always sends updated values associated with the current timestamp (Listing 14). Also, the service requires for the temperature and pressure values to be in Fahrenheit and inches, respectively. We apply conversion methods (from celsius and hectopascal) previously to uploading.

```
#!/usr/bin/python3
import os
import requests
import datetime
import subprocess
import time

#convert temperature degrees to fahrenheit
def convertCtoF(degrees):
  fah = float(degrees)*(1.8)+float(32)
  return fah

#convert pressure hpa to inches
def converthPatoInches(hPa):
  inch = float(hPa)*0.02953
  return inch

#calculate dew point from temperature(degrees) and humidity(percentage)
def calcDewPoint(temp,hum):
  temp = float(temp)
  hum = float(hum)
  dew = pow(hum/100,1/8)*(112+0.9*temp)+0.1*temp-112
  return dew

#send data to PBG
def uploadData(params):
  print(params)
  url="https://weatherstation.wunderground.com/weatherstation/updateweatherstation.php"
  try:
    req = requests.get(url,params=params)
    print(str(req.status_code) + " " + str(req.text))
  except requests.exceptions.RequestException as e:
    print("WUndergroud request error:",e)
```

Listing 13: Part of the Python script for uploading to Weather Underground.

The conversion formula used to convert the temperature values from Degrees (T_C) to Fahrenheit (T_F) (from [20]):

$$T_F = T_C \frac{9}{5} + 32$$

The conversion formula used to convert the atmospheric pressure values from hectopascal (P_{hPa}) to Inches (P_{Inches}) (from [34]):

$$P_{Inches} = P_{hPa} * 0.02953$$

To note that we opted for running a separate thread for the WUnderground, which reads the different data type values from the dump files and uploads them on a single request. The reason for that approach is because the web interface only displays values received from the latest request. If each capture thread would send a request, only the latest value (the one with the most recent timestamp) would appear, seeming it was the only information being sent.

```
{
    'ID': '*****',
    'PASSWORD': '*****',
    'dateutc': '2018-07-26 10:37:29',
    'action': 'updateraw',
    'baromin': '28.74',
    'temp2f': '70.93',
    'tempf': '71.97',
    'humidity': '79.17'
}
```

Listing 14: Example of parameters in a WUnderground upload request.

Chapter 6

Discussion

Before deploying our station at Parque Biológico de Gaia, we experimented it for a month at home, in an open patio. We had a chance to test all software features and troubleshoot bugs in the code. It was also an opportunity to check the viability of the physical structure. We concluded that, during the day, the covering board is able to fully protect the control box from the sun. It also rained for several days during that time and the structure did not suffer any water infiltration. Regarding temperatures, the CPU was able to stay at a maximum of about 55ºC at common ambient temperatures, with medium workload (it reached a peak of 67ºC when the ambient temperature was in the forties early in August). It is worth mentioning that the cover is critical for keeping low temperatures. Without it, at heat peak hours, the internal temperature of the box can reach critical temperatures (> 80ºC).

Ever since the EOS has been deployed at Parque Biológico de Gaia, it has been collecting data. The astronomy camera has been taking all-sky images during the astronomical twilight at a periodicity of 1800s (half an hour). All the meteorological components as well as, the luminosity sensor have been capturing 24/7 at the same cadence. Every half hour, the latest meteorological data is being uploaded to Weather Underground. From 31st of August to 5th of September, the station was offline. A power surge occurred in Parque Biológico de Gaia, which corrupted the system. Hence, it was necessary to re-install a cloned image of the system into the SD card.

6.1 Astronomy Camera

Next, we present some of the all-sky pictures taken with the ZWO ASI120MM camera using our capturing software. Figures 6.1 and 6.2 illustrate a rainy and cloudy morning in Vila Nova de Gaia during the experimentation stage on 29 of April.

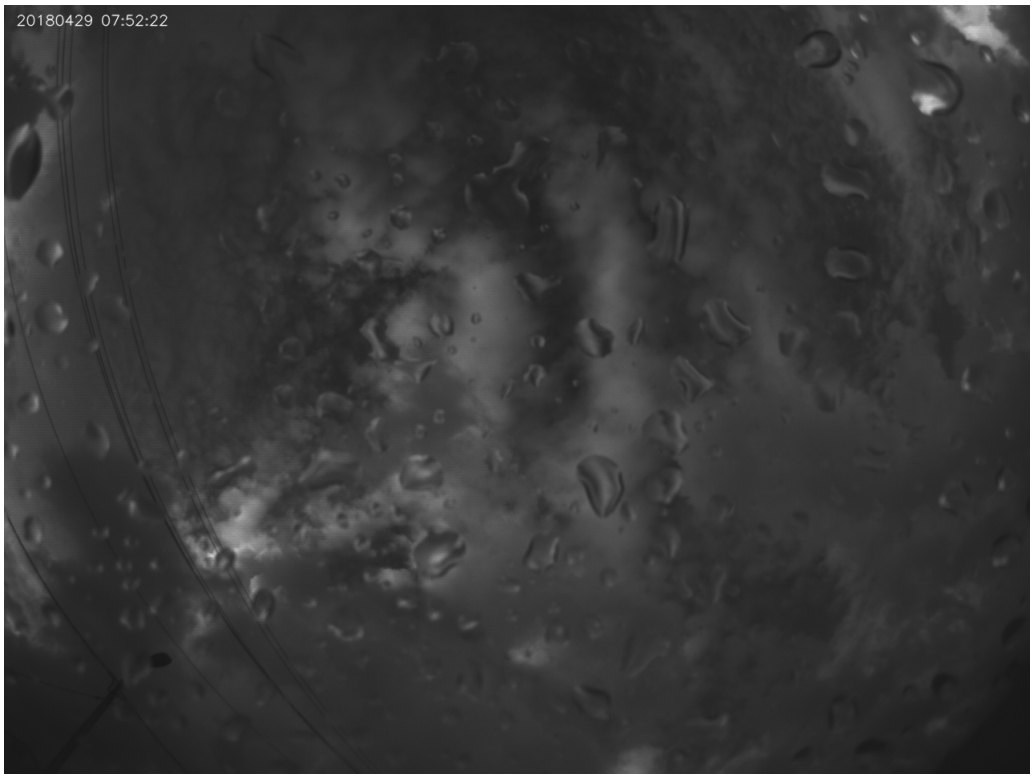


Figure 6.1: All-sky picture (29 of April at 07:52).



Figure 6.2: All-sky picture (29 of April at 09:03).

We verified that in cold days and sometimes during the night, the dome of the camera's support got wet, as a result of weather condensation. Still, as temperatures stabilized, it would dry itself without requiring manual cleaning. Since the location was within the city (in Vila Nova de Gaia, Porto), it was impossible to watch the stars clearly. Additionally, near the experimentation yard, there was a street light interfering with the night captures.

After the deployment of the EOS at the Parque Biológico de Gaia, we had the opportunity to collect more consistent results. The following figures illustrate all the different sky scenarios we were able to capture.

Figure 6.3 and Figure 6.4 are sky images captured during the day. For capturing during the day, the ZWO camera is configured with less than 1ms of exposure time (otherwise, the sensor gets saturated and the resulting pictures are completely white).

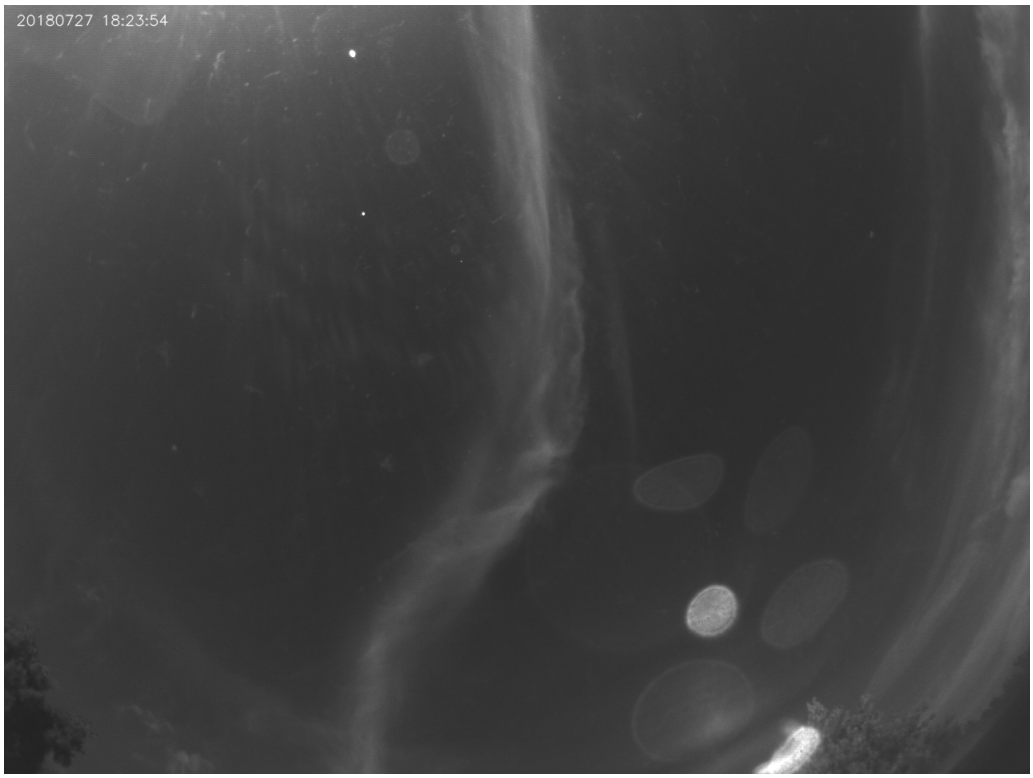


Figure 6.3: All-sky picture (27 of July at 18:23).



Figure 6.4: All-sky picture (28 of July at 12:12).

Parque Biológico de Gaia is a natural park with many trees, shrubs and animals at 1.85km from Rio Douro, which lead to a very damp environment. The station is surrounded with greenery and, many times, the external dome of the camera's enclosure condensates and inhibits the capturing of a clear sky. Also, the dome is made of acrylic, which does not conduct heat well and allows dew formation more easily. Usually, over a night, the first images are the ones which display increasingly more condensation (Figure 6.5) and, later on, the dome is eventually dried out, allowing more neat sky imaging. The drying process happens, when consistent wind rises in the region. Sometimes, whether precipitation occurs or there is lack of wind, the dome becomes wet to the point of affecting the entire viewing angle (Figure 6.6).

For the night captures, the camera was configured with 5s of exposure time and no gain.



Figure 6.5: All-sky picture (25 of July at 01:07).

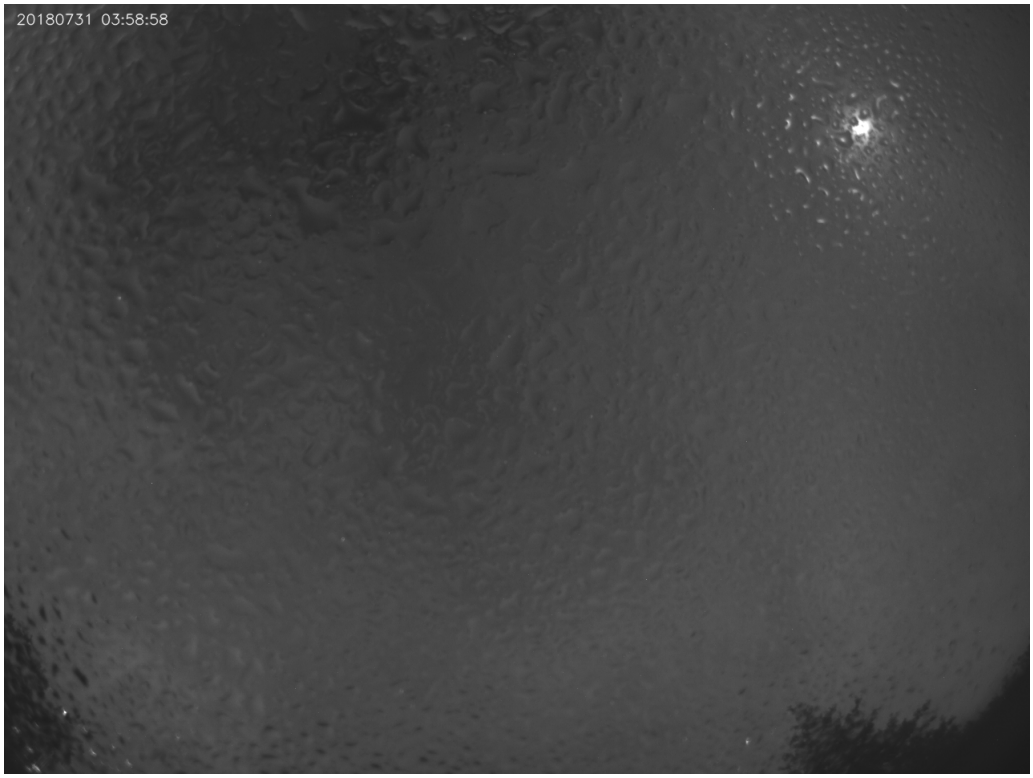


Figure 6.6: All-sky picture (31 of July at 03:58).

Despite the dominant condensation, we were able to capture some moistureless images as well. Thus, we can see a clear moon (Figure 6.7), clouds (Figure 6.8) and a few stars (Figure 6.9).



Figure 6.7: All-sky picture (5 of August at 04:20).



Figure 6.8: All-sky picture (8 of August at 01:14).



Figure 6.9: All-sky picture (20 of August at 04:23).

The camera's support was able to keep it at adequate temperatures during the day (approximately 30ºC at heat peak hours), by allowing a soft breeze from the bottom of the enclosure. Furthermore, it prevented the camera's dome from condensing during the night.

6.2 Temperature

Regarding temperature, we have been using two components: BME280 [12] and DS18B20 [27]. Both capture at the same instances with the same periodicity. Figure 6.10 illustrates a time series chart containing samples captured from both devices. As demonstrated, they are fairly equivalent. Both have similar accuracy specifications and price tags. As validation, we have compared the retrieved values with information provided by Instituto Português do Mar e da Atmosfera (IPMA) [81] for the same region and with nearby Weather Underground stations. The values are overall coherent, with occasional spurious readings. These isolated values can be related to heightened noise. In the first week of August, when the ambient temperatures were in the forties, a single BME280 sample reached 45ºC (at 14:07 on 2nd of August) - the maximum value measured yet, matching the absolute maximum measured in some districts of Portugal that day. During the hottest hours of that week, the temperature values were balanced at about 41ºC, though.

As previously stated, there is a gap between 31st of August and 5th of September. There was a power surge problem, which caused the station to be offline during that period.

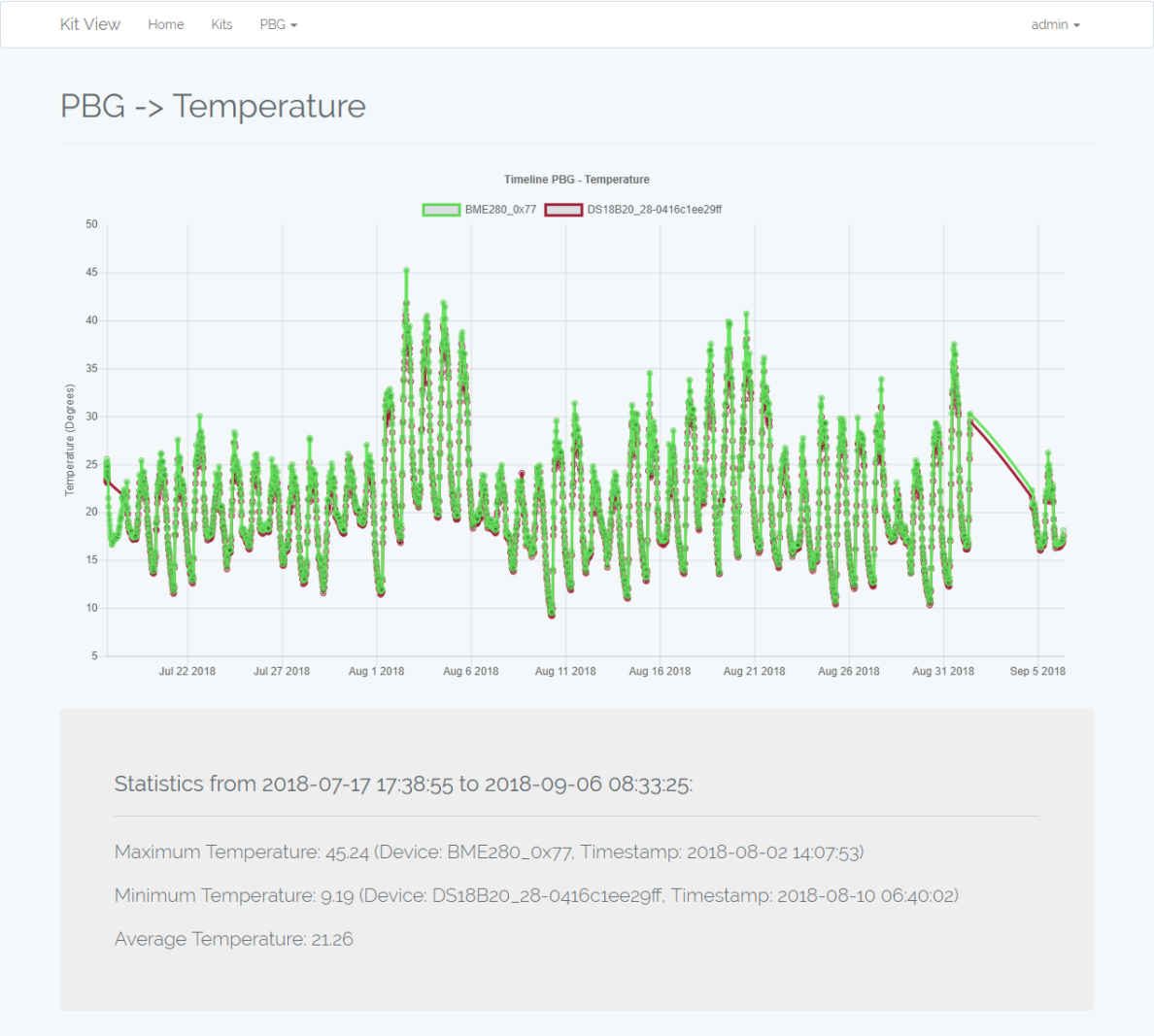


Figure 6.10: Temperature Chart.

The chart in Figure 6.11 illustrates the temperature difference between the two components over the first week of August. Overall, BME280 measures slightly higher values (no more than 4 degrees of difference). With lower temperatures, the discrepancy is also lower.

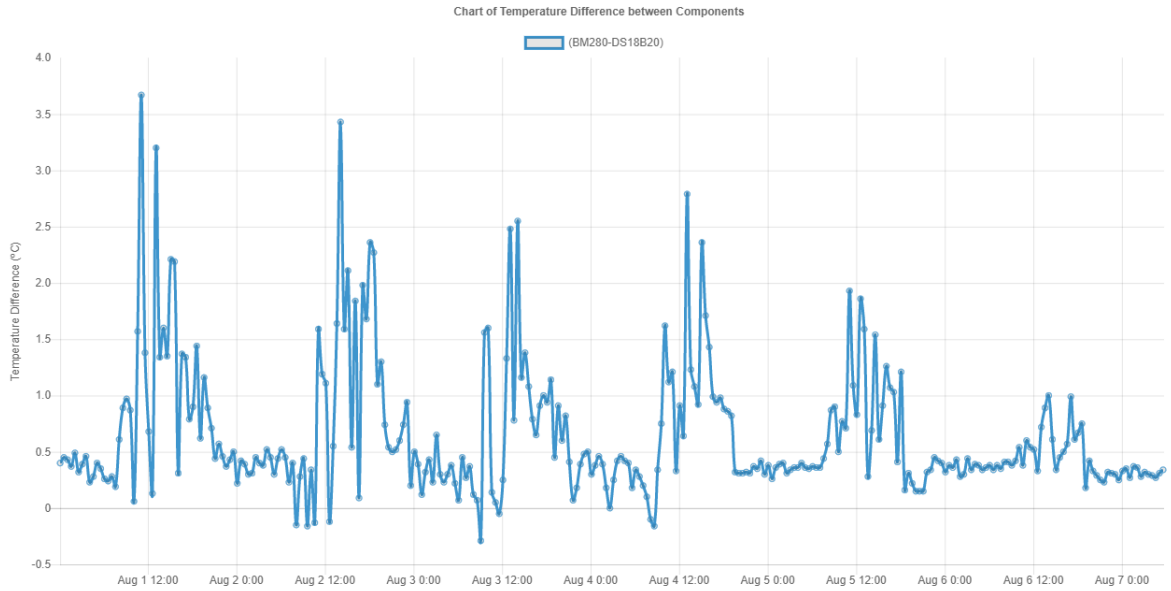


Figure 6.11: Temperature Difference Chart.

The scatter plot in Figure 6.12 represents the relationship between both components. The readings are clearly more dispersed at higher temperatures, with the BME280 typically capturing higher values compared to the DS18B20.

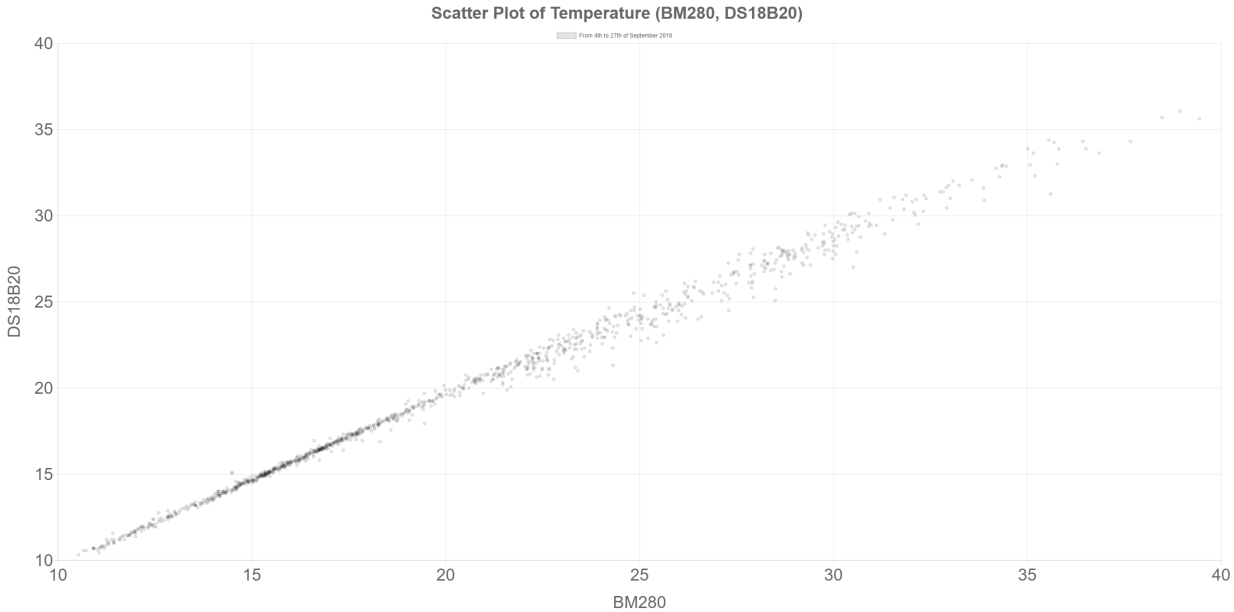


Figure 6.12: Scatter Plot of Temperature.

6.3 Humidity

The BME280 was also used to gather data related to humidity (Figure 6.13). Due to the humid location, humidity measurements were at 100% almost every night. The driest night occurred in 20th of August when it reached a peak of 93%. 25% was the minimum value achieved in 2nd of August (at 13:37), matching with the hottest week. Despite the captures taking place in the summer, most days were wet, resulting in an high overall humidity distribution. Consequently, the plastic cups serving as the radiation shield for the components have got a bit worn out. An understandable circumstance given the severe weather.



Figure 6.13: Humidity Chart.

6.4 Atmospheric Pressure

Regarding atmospheric pressure, it was used the BME280 device. The corresponding chart is displayed in Figure 6.14. Unfortunately, the values captured until the 27th of August are inaccurate. It was detected by Prof. Sérgio Crisóstomo, who warned that the values were deviating a few dozens hectopascal within a day, which is not possible. After a datasheet [12] analysis and a driver revision, we discovered the problem. In order to collect an accurate pressure measurement, the device needs to first obtain a temperature reading. A certain register in the device is set upon reading temperature (global variable t_fine, which stores fine temperature), which is then used to calculate pressure. A tweak in the driver to always perform a temperature reading before a pressure reading, solved the issue. In this manner, we are able to use the device for different data type captures independently. After applying the correction, the values started to be normal and steady, as they should.

Typically, reference atmospheric pressure values are based on the sea level. In order to be comparable with the data provided by IPMA [81], we applied the formula (from [57]):

$$P_0 = P(1 - \frac{0.0065h}{T + 0.0065h + 273.15})^{-5.257}$$

The altitude (h) of the station's location used in the calculation was 65m (retrieved from a topographic map [43]). P_0 refers to the corrected atmospheric pressure, P to the pressure measured by the sensor and T to the temperature reading.



Figure 6.14: Atmospheric Pressure Chart.

6.5 Luminosity

For luminosity, we used the TSL2561 [75] device. Although we use the device in the exterior, it is intended for inside light situations. Its maximum range value is specified at 40,000 Lux. Since a daylight can reach up to 120,000 Lux, many relative minimums frequently appear in hours of greater luminosity (Figure 6.15). It happens when the sunlight values are over 35,000 Lux, causing an overflow in the result. The only way to use the device in such circumstances is configured with minimum integration time (13ms) and no gain. Setting a greater integration time (101ms or 402ms) or adding gain causes a sensor saturation and a driver crash. For proper sunlight measuring, we must use a device approved for outdoor applications.

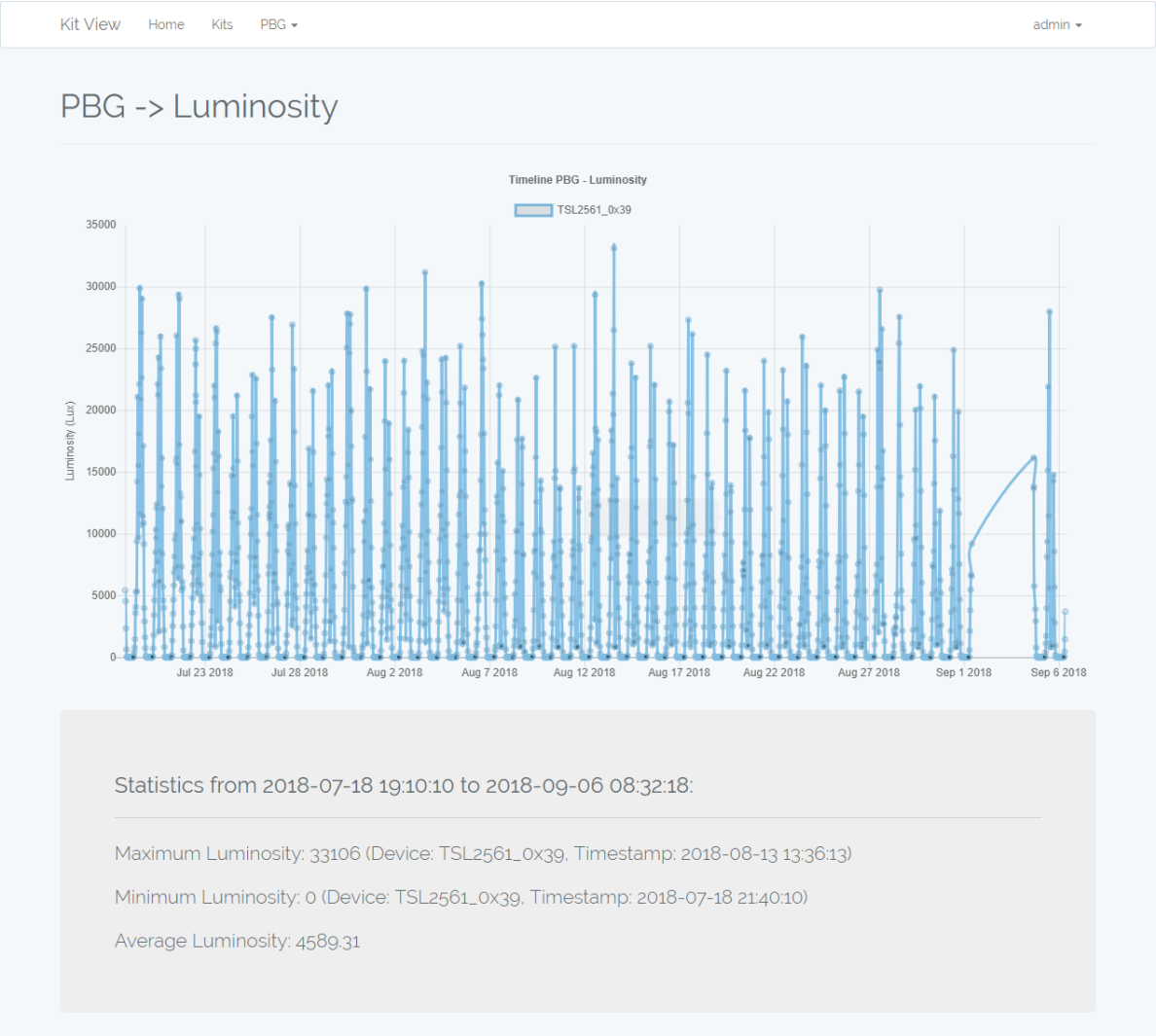


Figure 6.15: Luminosity Chart.

6.6 Weather Underground

The weather data captured by our prototype is also uploaded to Weather Underground. Figure 6.16 illustrates a screenshot of the corresponding WUnderground page [1].

[1] https://www.wunderground.com/personal-weather-station/dashboard?ID=ISANTAMA1762

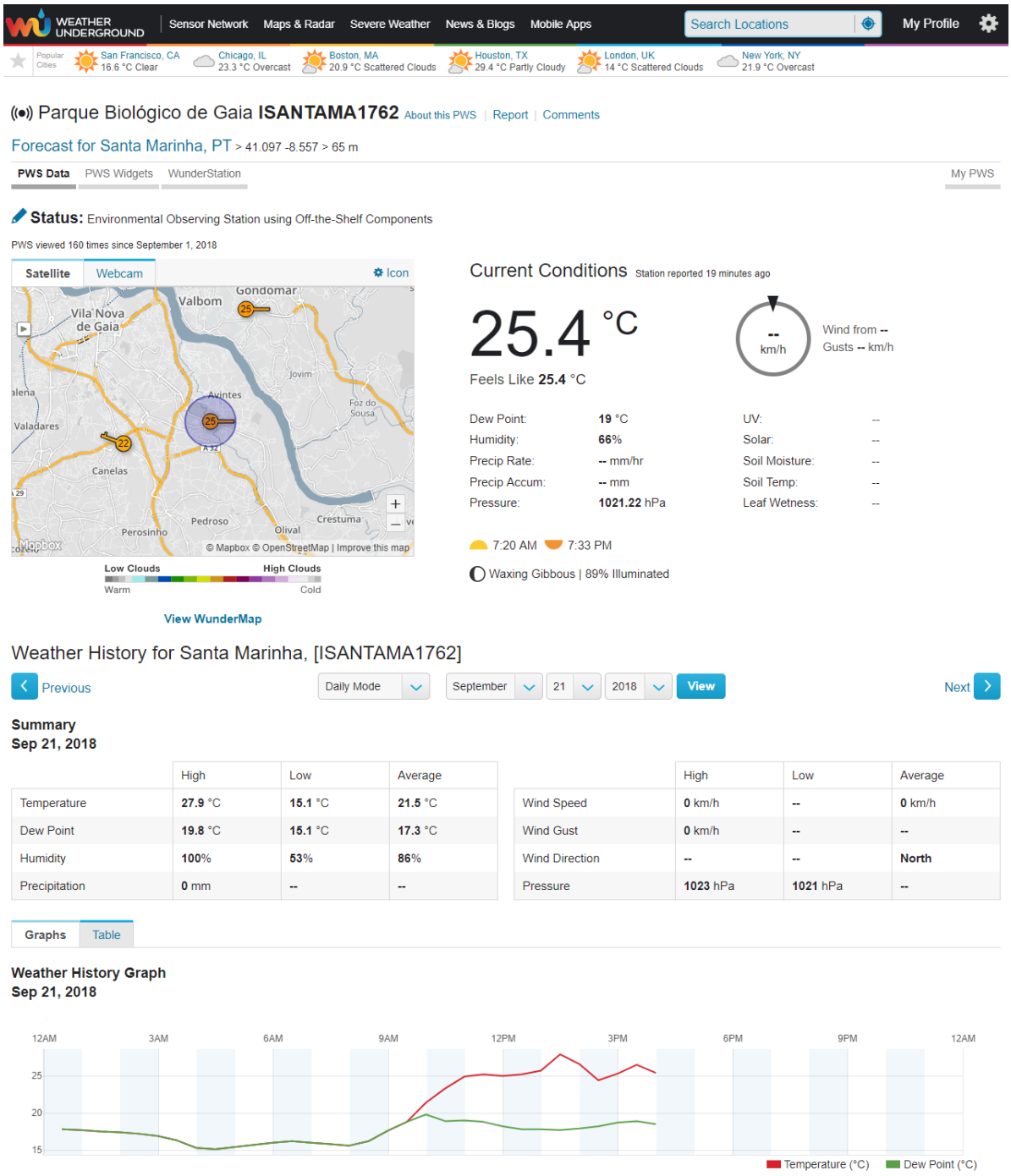


Figure 6.16: PBG - Weather Underground.

The station has been operating for enough time for us to collect a consistent amount of data and be able to draw conclusions. With a low-cost budget, we were able to build our prototype from scratch using off-the-shelf components. Furthermore, allowing a fully customization of its attached devices, it has the flexibility to be re-purposed for different environmental observing scenarios. For now, our prototype EOS meets our expectations, harvesting data related to the two initial intended schemes: astronomy and meteorology.

The weather has caused some level of deterioration in the radiation shield and it will need to be replaced in a recent future. The camera's enclosure is still in perfect conditions, because of the robustness of the PVC material. The structure's stand on top of the observatory was built with iron, so we coated it with primer paint to prevent it from corroding. Nonetheless, the EOS's physical structure will require periodic maintenance in order to continue protecting the internal components.

Chapter 7

Conclusions and Future Work

Technology has provided us with numerous mechanisms to develop and enhance technological applications. These days, we have available hardware equipment which allows us to create practical real-life projects at low-prices. Computing platforms, like the Raspberry Pi, and the number of off-the-shelf sensor components that can be easily acquired provide us tools to build complex embedded systems that are not hardware and software restricted to their initial scopes. To some extent, such systems can be re-purposed for different schemes. Regarding environmental observing, most of the existing commercial solutions (e.g., forecasting systems, air quality monitors) have narrow features and offer none/minimal customization options. The purpose of our work was to study how we could develop a modular and flexible system capable of functioning for different environmental areas. Moreover, the re-adjustment of the system should be as simple as possible, so a user with basic knowledge could be capable of performing it alone or with minimal aid. We also wanted to build a system with low-cost expenses and using off-the-shelf components.

After the implementation of this project, we feel a sense of accomplishment. We managed to create a system to support multiple Environmental Observing Stations. Our prototype was built from scratch using off-the shelf components and integrating devices for both initial intended scenarios: astronomy and meteorology. Creating a flexible system for a broad range of natural application fields implies the use of a wide variety of devices. Thus, it required us to research communication interfaces and protocols and discover a way to implement such a feature. Specifically, an EOS had to be able to constantly detect connected devices and run the software accordingly. Hence, the usage of addressable communication protocols (such as I^2C and One-Wire) offers a way of recognizing individual components and mapping their detected address to other related information. During the implementation stage, we felt an increasing sense of comfort and maturity. It resulted in better understanding our initial research, as well as implementing features we had not planned (e.g., developing the content insertion app). For programming the software, we had into consideration the target controllers (i.e., hardware constraints) and a good maintenance strategy (i.e., modular design, well organized code for adding/editing features in the future, scalability). The applications were developed using web

technologies, Java and Python programming languages and Linux utilities.

Along the way, we have also had minor setbacks. For our prototype, it would have been interesting to use more meteorological devices (such as, an anemometer and a rain gauge). Unfortunately, purchasing those kind of components would have exceeded the low-cost budget. Nevertheless, we designed our system to support the incorporation of those types of devices in the future. Another difficulty we encountered was determining the deployment location for our EOS. We intended to place it at a site where its purpose would be suitable. A perfectly example of such a place is Parque Biológico de Gaia. After contacting them and explaining in what the project consisted, we had the opportunity to deploy our prototype on top of the roof of their astronomical observatory. Hence, we were able to deploy our prototype in an environment related to its purposes. Regarding our astronomical results, we have had some difficulties getting proper all-sky images during the astronomical twilight. The very humid climate many times produces condensation on top of the structure's dome, not allowing the camera to take clear pictures. Light pollution was also an interference factor in monitoring the sky, particularly during the experimentation stage.

Our system was designed to support multiple EOS. As future work, we might deploy stations to observe more environmental scenarios. Our prototype allowed us to establish a base standard. The creation of its physical structure using off-the-shelf components and the results of its utilization have given us new ideas to adopt in the assembly of future EOS. Keeping a low-cost and a do-it-yourself perspective as goals, we are able to create unique and authentic stations to run our software. Additionally, it would be enriching to explore more alternatives regarding power and connectivity. Powering a station with renewable energy (integrating a solar panel, for example) would permit a station to be maintained online for long periods of time at isolated locations, in addiction to being environment-friendly. Also, connecting an EOS to a LPWAN would provide Internet connectivity, that would otherwise not be possible. Lastly, we conclude that there is always room for improvement with continuous development and experimentation.

References

[1] Adafruit. I2C-Bus Addresses Table. https://cdn-learn.adafruit.com/downloads/pdf/i2c-addresses.pdf. Accessed: April-2018.

[2] AirPi. http://airpi.es/index.php. Accessed: November-2017.

[3] Anemometer Wind Speed Sensor. https://www.adafruit.com/product/1733. Accessed: November-2017.

[4] Anemometers (Ultrasonic and Mechanical). https://www.lufft.com/blog/app/uploads/2016/05/Title-Anemometers.jpg. Accessed: December-2017.

[5] Arduino. https://www.arduino.cc/. Accessed: November-2017.

[6] Arduino Weather Station Project. http://cactus.io/projects/weather/arduino-weather-station. Accessed December-2017.

[7] Arduino Uno. https://store.arduino.cc/usa/arduino-uno-rev3. Accessed: November-2017.

[8] ZWO ASI120MM Astronomy Camera. https://astronomy-imaging-camera.com/product/asi120mm. Accessed: December-2017.

[9] ZWO ASI SDK. https://astronomy-imaging-camera.com/software-drivers. Accessed: November-2017.

[10] BeagleBoard. https://beagleboard.org. Accessed: November-2017.

[11] BeagleBoard Black. https://beagleboard.org/black. Accessed: November-2017.

[12] BME280 Sensor. https://cdn-shop.adafruit.com/datasheets/BST-BME280_DS001-10.pdf. Accessed: March-2018.

[13] BME280 Sensor Image. https://www.modmypi.com/raspberry-pi/sensors-1061/temperaturepressurehumidity-1066/adafruit-bme280-i2c-or-spi-temperature-humidity-pressure-sensor. Accessed: March-2018.

[14] Bootstrap. https://getbootstrap.com/. Accessed: April-2018.

[15] Chart.js. https://www.chartjs.org/. Accessed: May-2018.

[16] Axis Communications. CCD and CMOS Sensor Technology. https://www.axis.com/files/whitepaper/wp_ccd_cmos_40722_en_1010_lo.pdf, 2010. Accessed: December-2017.

[17] Davis Instruments Vantage PRO2 Image. https://www.scientificsales.com/Davis-6152-Wireless-Vantage-Pro2-p/6152.htm. Accessed: April-2018.

[18] Davis Instruments Vantage VUE Image. https://www.scientificsales.com/Davis-6250-Vantage-Vue-Wireless-Weather-Station-p/6250.htm. Accessed: April-2018.

[19] GSM SMS Remote Control Relay Switch. https://www.ebay.com/itm/GSM-SMS-Remote-Control-Relay-Switch-Phone-Smart-Wireless-Remote-Controller-US-EU-/222294323939. Accessed: December-2017.

[20] Celsius to Fahrenheit Conversion. https://www.rapidtables.com/convert/temperature/celsius-to-fahrenheit.html. Accessed: July-2018.

[21] Dew Point Equations Formulas Calculator. https://www.ajdesigner.com/phphumidity/dewpoint_equation_dewpoint_temperature.php. Accessed: August-2018.

[22] DHT Temperature and Humidity Sensors. https://learn.adafruit.com/dht. Accessed: November-2017.

[23] Digi XBee Embedded Cellular Modems. https://www.digi.com/products/xbee-rf-solutions/embedded-cellular-modems. Accessed: December-2017.

[24] Digital Barometric Air Pressure (BAP) Sensor (SM1131). https://www.si-micro.com/products/digital-systems/barometric-air-pressure-bap-sensor-sm1131.html. Accessed: November-2017.

[25] Digital UV Index/IR/Visible Light Sensor. https://www.silabs.com/documents/public/data-sheets/Si1145-46-47.pdf. Accessed: December-2017.

[26] Digital Rain Gauge Sensor. https://www.raspberrypi.org/learning/sensing-the-weather/guides/rain_gauge/. Accessed: December-2017.

[27] DS18B20 Temperature and Humidity Sensor. https://cdn-shop.adafruit.com/datasheets/DS18B20.pdf. Accessed: December-2017.

[28] Extended Coverage – GSM – Internet of Things (EC-GSM-IoT). https://www.gsma.com/iot/extended-coverage-gsm-internet-of-things-ec-gsm-iot/. Accessed: January-2018.

[29] Github - A Raspberry Pi operated Wireless Allsky Camera. https://github.com/thomasjacquin/allsky. Accessed: November-2017.

[30] MTK3339 GPS Module. https://www.adafruit.com/product/746. Accessed: January-2018.

[31] GSMA. 3GPP Low Power Wide Area Technologies - White Paper. https://www.gsma.com/iot/wp-content/uploads/2016/10/3GPP-Low-Power-Wide-Area-Technologies-GSMA-White-Paper.pdf. Accessed: January-2018.

[32] DIY DC 6-35V to 5V 3A Double USB Voltage Step Down Regulator Module. https://www.gearbest.com/development-boards/pp_69337.html. Accessed: December-2017.

[33] Giles Harrison. *Meteorological Measurements and Instrumentation*. Wiley-Blackwell, 2014. Advancing Weather and Climate Science Series.

[34] Convert hPa to Inch of Mercury - Conversion of Measurement Units. https://www.convertunits.com/from/hPa/to/inch+of+mercury. Accessed: July-2018.

[35] I2C-Bus Architecture. http://www.handsonembedded.com/stm32f103-spl-tutorial-6/. Accessed: May-2018.

[36] Introduction to I2C and SPI Protocols. https://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/. Accessed: May-2018.

[37] Tomas Jacquin. Wireless All-Sky Camera Project. http://www.instructables.com/id/Wireless-All-Sky-Camera/. Accessed: October-2017.

[38] JSON Parser for C++. https://github.com/nlohmann/json. Accessed: April-2018.

[39] Laravel - The PHP Framework For Web Artisans. https://laravel.com/. Accessed: June-2018.

[40] LoRa Shields. https://www.cooking-hacks.com/documentation/tutorials/extreme-range-lora-sx1272-module-shield-arduino-raspberry-pi-intel-galileo/. Accessed: January-2018.

[41] LPWAN - Licensed vs. Unlicensed. https://www.korewireless.com/blog/lpwan-licensed-vs.-unlicensed. Accessed: January-2018.

[42] LTE-M. https://www.link-labs.com/blog/what-is-lte-m. Accessed: January-2018.

[43] Mapa Topográfico - Vila Nova de Gaia. http://pt-pt.topographic-map.com/places/Vila-Nova-de-Gaia-7890785/. Accessed: August-2018.

[44] MCP3008 - 8-Channel 10-Bit ADC. https://www.adafruit.com/product/856. Accessed: November-2017.

[45] MCP9808 High Accuracy I2C Temperature Sensor. https://www.adafruit.com/product/1782. Accessed: November-2017.

[46] MPL3115A2 - I2C Barometric Pressure/Altitude/Temperature Sensor. https://cdn.sparkfun.com/datasheets/Sensors/Pressure/MPL3115A2.pdf. Accessed: November-2017.

[47] What is Model-View-Controller? https://blog.cloudboost.io/what-is-model-view-controller-124a9942246. Accessed: June-2018.

[48] Oculus All-Sky Camera. https://www.sxccd.com/oculus-all-sky-camera. Accessed: November-2017.

[49] OMEA All-Sky Camera. http://www.alcor-system.com/new/AllSky/Omea_camera.html. Accessed: December-2017.

[50] One-Wire Bus. http://www.python-exemplary.com/index_en.php?inhalt_links=navigation_en.inc.php&inhalt_mitte=raspi/en/temperature.inc.php. Accessed: May-2018.

[51] Overview of 1-Wire Technology and Its Use. https://www.maximintegrated.com/en/app-notes/index.mvp/id/1796. Accessed: May-2018.

[52] OpenCV Library. https://opencv.org/. Accessed: November-2017.

[53] M. Popa and C. Iapa. Embedded Weather Station with Remote Wireless Control. In *Telecommunications Forum (TELFOR), 2011 19th*, pages 297–300. IEEE, 2011.

[54] Original Xiaomi Power Bank 2C. https://www.gearbest.com/power-banks/pp_798193.html. Accessed: December-2017.

[55] Poweradd Apollo 7200mAh Solar Panel Power-bank. http://www.ebay.com/gds/Top-10-Solar-Power-Banks-/10000000204826081/g.html. Accessed: December-2017.

[56] What is the Difference Between 18650 and Li-polymer Battery? https://www.easyacc.com/media-center/what-is-the-difference-between-18650-and-li-polymer-battery/. Accessed: December-2017.

[57] Conversion to Sea-Level Pressure Calculator. https://keisan.casio.com/exec/system/1224575267. Accessed: August-2018.

[58] Raspberry Pi. https://www.raspberrypi.org/. Accessed: November-2017.

[59] Raspberry Pi 3 Model B. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. Accessed: November-2017.

[60] Raspberry Pi Weather Station. https://www.raspberrypi.org/learning/weather-station-guide/. Accessed: November-2017.

[61] Raspberry Weather. https://www.raspberryweather.com/necessary-parts/. Accessed: November-2017.

[62] Real-Time Operating System. https://www.freertos.org/about-RTOS.html. Accessed: May-2018.

[63] P. Sankar and S. R. Norman. Embedded System for Monitoring Atmospheric Weather Conditions using Weather Balloon. In *In International Conference on Control, Automation, Communication and Energy Conservation (INCACEC 2009)*, pages 1–4. IEEE, 2009.

[64] NXP Semiconductors. I2C-Bus Specification. https://www.nxp.com/docs/en/user-guide/UM10204.pdf. Accessed: April-2018.

[65] Sensor Platform for Gas Detection (PR1209). https://www.micronas.com/en/news-events/trade-news/micronas-presents-gas-86xyb-%E2%80%93-smallest-and-most-efficient-multi-parametric. Accessed: November-2017.

[66] SHT1x Humidity and Temperature Sensor IC. https://cdn.sparkfun.com/datasheets/Sensors/Pressure/Sensirion_Humidity_SHT1x_Datasheet_V5.pdf. Accessed: November-2017.

[67] Sigfox Network Architecture. http://www.rfwireless-world.com/Tutorials/Sigfox-network-architecture.html. Accessed: January-2018.

[68] Sigfox Shields. https://www.cooking-hacks.com/documentation/tutorials/sigfox-connectivity-arduino-raspberry-pi-868mhz-europe-900mhz-us/. Accessed: January-2018.

[69] R. S. Sinha, Y. Wei, and S. Hwang. A Survey on LPWA Technology: LoRa and NB-IoT. *ICT Express*, 2017.

[70] Solar Home System. http://www.leonics.com/system/solar_photovoltaic/solar_home_system/solar_home_system_en.php. Accessed: December-2017.

[71] SPI-bus Architecture. http://dlnware.com/theory/SPI-Bus. Accessed: May-2018.

[72] Serial Peripheral Interface (SPI). https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi. Accessed: May-2018.

[73] Sunwait. http://www.risacher.org/sunwait/. Accessed: February-2018.

[74] TGS 2600 - for the Detection of Air Contaminants. http://www.figarosensor.com/products/2600pdf.pdf. Accessed: November-2017.

[75] TSL2561 Digital Luminosity Sensor. https://cdn-learn.adafruit.com/downloads/pdf/tsl2561.pdf. Accessed: December-2017.

[76] Ultrasonic Anemometer. https://soldernerd.com/arduino-ultrasonic-anemometer/. Accessed: November-2017.

[77] UPS PIco - Uninterruptible Power Supply. https://www.modmypi.com/raspberry-pi/power-1051/ups-boards-1051/ups-pico. Accessed: August-2018.

[78] USB Pinout. http://pinouts.ru/Slots/USB_pinout.shtml. Accessed: May-2018.

[79] Serial Communication Protocols: The Basics. https://www.totalphase.com/blog/2017/08/serial-communication-protocols-the-basics/. Accessed: May-2018.

[80] USB Data Transfer Types. http://www.jungo.com/st/support/documentation/windriver/10.2.0/wdusb_manual.mhtml/USB_data_transfer_types.html. Accessed: May-2018.

[81] IPMA Observação de Superfície - Rede de Estações Meteorológicas. https://www.ipma.pt/pt/otempo/obs.superficie/. Accessed: July-2018.

[82] Weather Board for the Raspberry Pi / Arduino – Grove. http://www.switchdoc.com/weatherboard/. Accessed: December-2017.

[83] World Meteorological Organization (WMO). *Guide to Meteorological Instruments and Methods of Observation*. WMO, 2014.

[84] WUnderground PWS - Upload Protocol. http://wiki.wunderground.com/index.php/PWS_-_Upload_Protocol. Accessed: May-2018.

[85] X. Xiong, K. Zheng, R. Xu, W. Xiang, and P. Chatzimisios. Low Power Wide Area Machine-to-Machine Networks: Key Techniques and Prototype. *IEEE Communications Magazine*, 53(9):64–71, 2015.