UNIVERSITY OF GRONINGEN

Healthy Offices
Bachelor Computing Science

Authors:
Sybren GJALTEMA (S2550083)
Timo SMIT (S2337789)

Supervisors:
Faris NIZAMIC
Marco AIELLO

Additional supervisor: Brian Setz
BSc project coordinator: Alexandru C. Telea

July 14, 2016

ABSTRACT

At the moment, our offices do not come across as very smart, nor is it clear how healthy they are. They waste energy by not acting according to our needs [1] and are completely oblivious to the working environment. With the help of small computers and sensor sets, the environment of a room can be measured. On this data, statistical analysis may be applied to give a relevant recommendation of what action to take to make the environment meet certain quality standards.

Hardware has developed in such a way that it is relatively cheap to purchase a compact pocket-size computer, to attach some sensors to it and to deploy one set in every room of a building. Using a Raspberry Pi and the GrovePi+ sensor set, we created such a setup with the help of the people at Sustainable Buildings [2], and deployed it at a couple of sample locations. We detail how we have created this setup, where we have deployed them, what data we gather from them and how we interpret this data.

A key part in understanding the metrics coming out of the sensors is to understand what acceptable ranges of values are for such a sensor. This thesis summarizes what values from sensors are considered ”healthy”, or, if healthiness is not affected by the metric, at the very least ”comfortable”.

CONTENTS

GLOSSARY

ACRONYMS

API application programming interface

BMS building management system

DS Distributed Systems

LDR light detecting resistor

GPIO general purpose I/O

PIR passive infrared

REST representational state transfer

RH relative humidity

RUG Rijksuniversiteit Groningen

CO_2 carbondioxide

UNITS

dB decibels

lx lux

mb millibar

ppm particles per million

hPa hectopascal

Part I

THE HEALTHY PI PROJECT

1

INTRODUCTION

It is not uncommon to work at a company and sit in the same room for most of the day. It is best for both the company as well as the employee if such a room is healthy and energy efficient. It is reported that ”up to 30 percent of new and remodeled buildings worldwide may be subject of excessive complains related to indoor air quality” [3]. These problems appear to arise more often when ”a building is operated or maintained in a manner that is inconsistent with its original design or prescribed operating procedures” [3].

As stated in the article, there may be several causes for a building to be ”sick”. One of the most important factor responsible for causing such a sick building, commonly referred to as SBS (Sick Building Syndrome), is inadequate ventilation or inappropriate lighting with absence of sunlight.

While it may be clear how much sunlight flows into a room, it is much harder to sense if the ventilation is adequate. If the ventilation is not adequate, you only find out after you get the symptoms (headache, dizziness, nausea, loss of concentration, ...) [3]. It is evidently important to keep buildings properly ventilated.

Increasing air quality can be done in a couple of different ways, like opening a window or increasing the amount of ventilation in a room. This may lead to extra energy consumption, however (heating leaving the building due to open window or simply extra power for the ventilation), so it would be better if these measurements would only be taken when necessary.

In this thesis we will answer the following research question:

How does one determine the healthiness of an office space based on sensory data? (1)

To answer this question in a better way, we have subdivided the problem into several smaller and easier to answer research questions:

Is it possible to determine the condition of an office space with a small set of sensors, namely CO_2, temperature, humidity and light sensors? (2)

How should the aforementioned sensors be used? (3)

How do the values of the sensors relate to the healthiness of a room? (4)

Is the collected data sufficiently significant and comparable? (5)

The Healthy Pi is a little box, developed for and with the people from Sustainable Buildings[1], consisting of a Raspberry Pi[2], a GrovePi+[3] and four different types of sensors. It gives insights into the healthiness of a room, giving its occupants and managers an idea of what can be improved in that environment.
An example would be that a CO_2 sensor senses a high concentration of CO_2 particles in the air. This may indicate that the room is improperly ventilated.

One of the goals of the Healthy Pi is to get better insights into what kind of an environment a room has, and how it can be improved as to increase productivity, healthiness and energy-efficiency. The sensors related to this project are sensors for measurement of levels of CO_2, light levels, temperature and humidity and movement. Using this data, we can get an idea of the state of a room.

A secondary goal is to make the Healthy Pi future-proof by making it as extensible as possible. The GrovePi definitely helps achieve this goal, due to its modular design (which you'll read more about in chapter 4). The framework we use for the code (as described in chapter 3) helped making the code modular, since it uses the Cake pattern [4]. We won't go into this intensely though, as it is a secondary goal.

The thesis is set up in the following way: we first look at similar projects in chapter 2. We then continue by giving a detailed description of the project setup, with chapter 3 going into the framework that we use and chapter 4 giving a look into how the sensors work. In chapters 6, 7 and 8, we list the results of our research and in chapter 9 we draw conclusions from said research. We follow it up by suggesting future work in chapter 10. Finally, we explain our methodology in chapter 11. Appendices can be found after these final chapters.

1 http://www.sustainablebuildings.nl/
2 https://www.raspberrypi.org/
3 http://www.dexterindustries.com/grovepi/

2

RELATED WORK

We were not the first team to gather environmental data from rooms. Other companies have developed similar technologies. In this chapter we discuss similar projects and how they are related to our project. We chose to only include devices that include sensors such as our own, as to keep the level of similarity close. The minimal set of sensors we define as CO_2, temperature and humidity.

2.1 DEXTER INDUSTRIES PYTHON SCRIPTS

The sensors we use for our setup are Seeedstudio [5] sensors. Each sensor has its own wiki page detailing the technical details of the product, as well as providing some example code (usually in the C programming language). The hardware specifications are also included, usually as a PDF download. Those PDF files contain roughly the same information as the wiki, but specified in more detail with additional technical details that most users won't need (such as the design of the hardware).

Furthermore, the GrovePi GitHub repository [6] contains a substantial amount of python example scripts that interface with the sensors as long as they are plugged in into the correct socket on the GrovePi. Using both examples, it was relatively easy to port the given example code to Scala.

For the sake of showing how the example code helped us, we will present you with the given code of the CO_2 sensor in appendix A.1. An instance that helped us understand how to use the sensor can be found on line number 8, where the command to get data from the sensor is defined. Furthermore, combining line numbers 17-20 shows us exactly how to calculate the ppm value that the sensor is giving us.
Please note that this was also specified in the PDF belonging to the CO_2 sensor, but having a working code-example makes it much easier for us to confirm that the code we have written performs nominally.

Seeedstudio makes over a hundred different types of sensors [7], of which we only use a few. Later we will see that using a GrovePi makes our project much more extensible (Chapter 10.3).

2.2 NETATMO

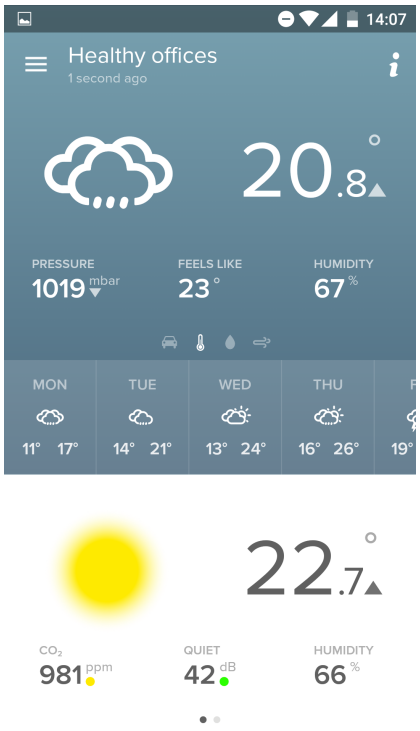

Figure 1.: Overview of the Netatmo app

Another company that does something very similar is Netatmo [8]. Their product is a weather station consisting of an indoor module and an outdoor module, with the option to buy additional modules such as a rain gauge and a wind gauge. Since we are looking to measure indoor environmental data only, we only needed the indoor module. We used a Netatmo weather station to compare the data from our sensors with the data from their sensors. The data generated by Netatmo indoor module includes, but is not limited to, the following types of data:

- Temperature in Celsius
- Humidity in percentages
- Air pressure in mb
- CO_2 levels in ppm
- Sound meter in dB

The outdoor module would add additional data regarding outdoor temperature and humidity, but, as stated before, this is irrelevant for our project and we have therefore not used the outdoor module.

Figure 1 is a screenshot of the application (v2.2.2) made by the Netatmo team to view the data collected by the weather station. As you can see, it shows the current situation of the room, as well as a general assessment of the room (the yellow dot). Furthermore, it shows the weather forecast and has a couple of other neat features.

The Netatmo website [8] states that CO_2 concentrations of 425 ppm are considered ”Very good”, values of 1180 ppm are considered ”Average” and values of 2250 ppm are considered ”Warning”. These values can be found back in the app indicate by a green, an orange or a red dot respectively. Please note that the station was placed right next to the computer monitor, so slightly elevated CO_2 levels are to be expected, as the user of the computer practically breathes into the sensor. We will later see that this had no effect on our comparison methods.

2.3 ARCHOS WEATHER STATION

The Archos Weather Station [9] is a weather station very similar to the Netatmo weather station. It comes with three different modules; an indoor module, an outdoor module and a soil sensor. The outdoor- and the soil modules are not very interesting, as they only

measure the outdoor- or soil temperature and humidity. Therefore, we're focusing on the indoor module, as it is most related to our project.

A neat feature of this particular weather station is that the indoor module can show you the air quality on request - no need to open any app. By simply passing your hand over the module it will show you the air quality by turning on its light. Green reflects a good air quality (which they define as having less than 1000 ppm CO_2), orange is reasonable (1000-2500 ppm CO_2) and, of course, red is bad (over 2500 ppm CO_2).

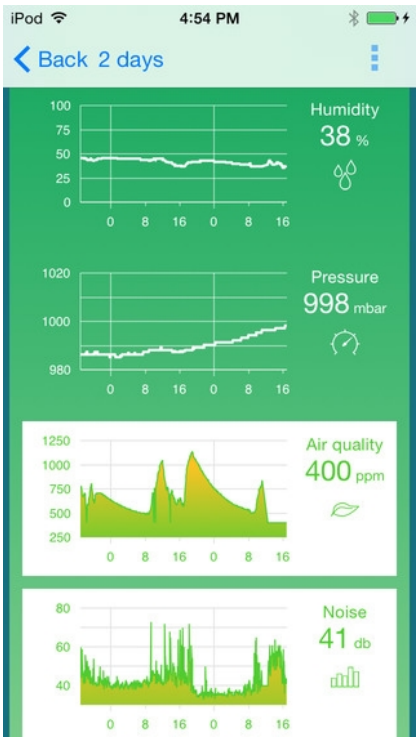

Figure 2.: Overview of the Archos app

As you can see in figure 2, the application is relatively similar to the Netatmo application. This application visualises every metric it can visualise, using color-shading to indicate the quality of the metric (if applicable).

The data generated by the Archos weather station includes, but is not limited to, the following types of data:

- Temperature in Celsius
- Humidity in percentages
- Air pressure in hPa
- CO_2 levels in ppm
- Sound meter in dB

As you can see, it measures exactly the same as the Netatmo device (note that $1hPa = 1mb$).

2.4 SUMMARY

We have found two products matching our criteria and one project that supplies the software for related hardware, with which you can build such a device yourself. We will compare the two devices to see what sets them apart from each other.

Both the Netatmo as well as the Archos weather stations use the same set of sensors. The only difference can be found in the outdoor or additional modules, which are not relevant to this project. They do, however, differ in regards to what concentrations of CO_2 they find "Good" or not. The Netatmo device considers values ranging from 425-800 ppm as "Very good", while the Archos device considers values under 1000 ppm as "Very good". Similarly, values up to 1700 are "Average" for the Netatmo, while the Archos considers 2500 ppm to be "Average". As you can see, these values lie quite far apart. Later in the thesis we will research what values we consider "Very good", "Average" or "Bad".

Part II

PROJECT SETUP

3

FRAMEWORK

The RUG Distributed Systems research group [2] supplied us with a framework that they created. It is written in Scala, a language built on top of the Java Virtual Machine (JVM). This framework is mainly used (in our case) to publish sensory data to a RabbitMQ server. Besides that, it comes with several utility libraries and functions that allow one to easily write code for sensor applications. This will be discussed in one of the following paragraphs.

The framework (see figure 3) consists of several modules that pass information in the form of a pipeline. Data is collected through sensors, which is in turn captured using an application. This application will push the data to a queue using a gateway service - it should in no way modify the data to maintain integrity. The gateway service is also used to control actuators (this is actually one of the connections that is two-way). The queue is automatically pulled (emptied) by a collector service, which will store the data in a Cassandra database. The accumulated data can then be retrieved using a representational state transfer (REST) application programming interface (API), allowing third-party applications to hook into the system.

Since the scope of our project is limited to sensors and collecting data, we have mostly used the gateway service and written our own sensor back-end. However, given the fact that we were asked to give a user feedback about the current environment, there is a small service available that connects with the REST service on the other end of the pipeline.

Finally, components that register the sensors and actuators, and services are planned (hence stylized in yellow boxes), but are not used in our project.

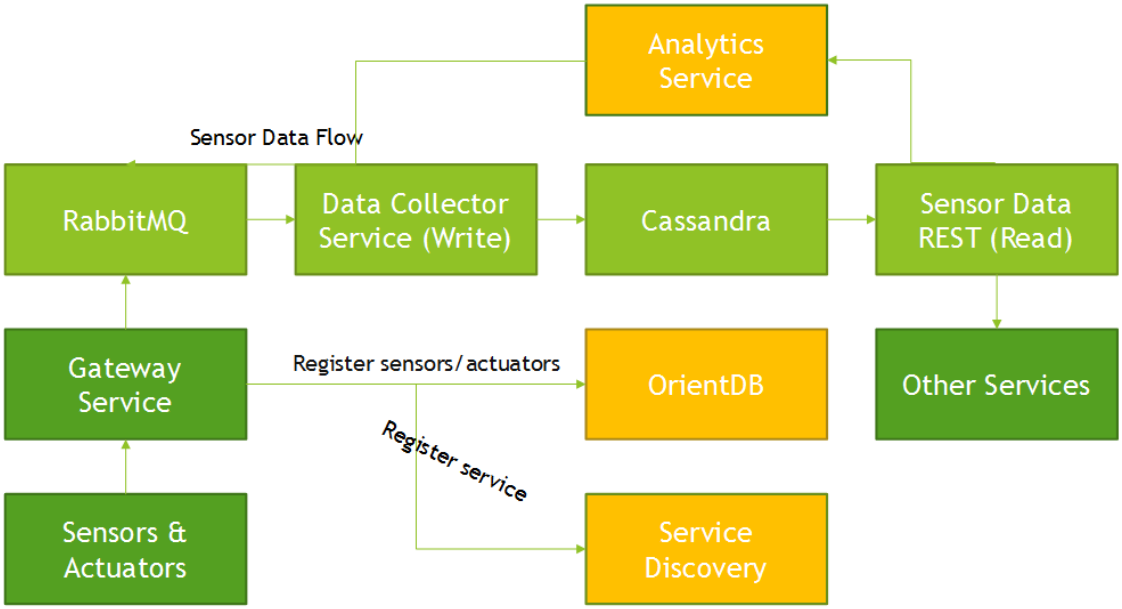

Figure 3.: Setup of the RUG DS framework

4

GROVEPI

Sensors are plugged into an interface called GrovePi, which is developed by Dexter Industries. This interface is connected as a whole to the Raspberry Pi by using the general purpose I/O (GPIO) pins on top of a Pi. The GrovePi allows for both analog as well as digital communication - internally they are both digital.

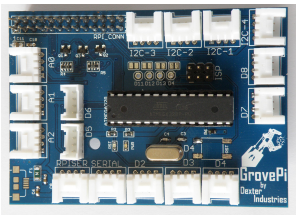

Figure 4.: GrovePi

4.1 GPIO AND I2C

Communication with regular analog and digital sensors is possible using the I2C protocol. This is done over a special bus (two of the GPIO pins, to be exact) to which each sensor is connected. One can distinguish between the sensors using their address on the bus (a pin identifier).

Reading a value from a sensor is a two-step process. One first has to request data, after which the data will become available on the bus. The request is an array consisting of four bytes: a command to send to the sensor, the address, and two bytes specific to the sensor (i.e. a model number). These latter bytes may also remain unused. To collect the data, one may simply read the data from the bus using the I2C protocol. Most devices will return an array of also four bytes, of which the middle two contain the value (in little endian format), and the last byte is a parity byte (checksum).

In addition to the I2C protocol, the GrovePi also comes with several other ports. One of these is the serial port, which allows one to directly communicate with a sensor that is plugged into the GrovePi. Via this port, the I2C bus is omitted and the sensor will be treated as a separate device in Unix-based systems.

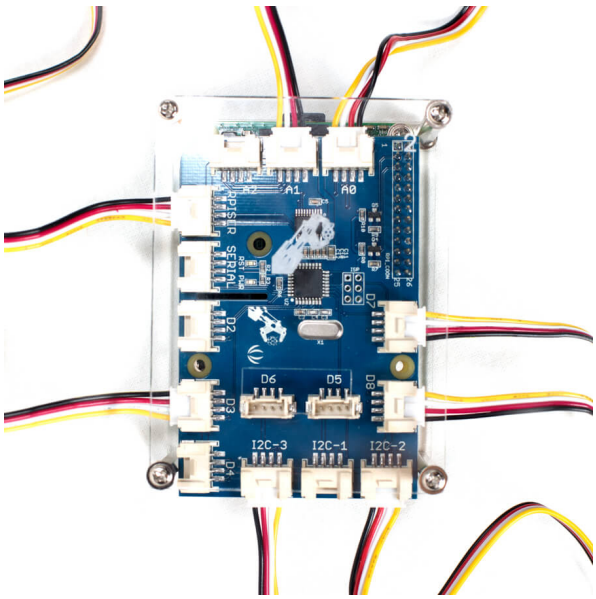

Figure 5.: Layout of a GrovePi

4.2 SENSORS

The Healthy Pi uses four different sensors; a CO_2 sensor, a temperature and humidity sensor, a light sensor and a motion sensor. Each of these sensors output different kinds of data. To make it clear what kind of data the sensor outputs, see the following table:

Sensor	Model	Data unit
CO_2	MH-Z16	particles per million (ppm)
DHT	DHT11	Celsius relative humidity (RH) (%)
Light	GL5528	lux (lx)
PIR	BISS0001	boolean

Table 1.: Data units outlined per sensor

In the following sections we will describe each of the sensors individually, illustrating what they are used for, what data they produce and how this data should be interpreted.

4.2.1 CO_2

The sensor we use for our project to measure the ppm value of CO_2 in a room is the MH-Z16 Seeedstudio Grove sensor [10]. The documentation for this sensor is quite complete. The creators of the sensor published the datasheet [11], containing relevant information such as how the sensor is built up, what data it expects, what data it outputs, on what kind of baud rate it runs, and much more.

Figure 6.: CO_2 sensor

4.2.1.1 *Setting up the sensor*

The sensor has a serial interface, and is to be connected to one of the serial ports of the GrovePi. We connected it to the `rpiserial` port, as this port is directly accessible on the Raspberry Pi. We could have chosen for the `serial` port, but the drawback of doing that is that you have to address the GrovePi before being able to transmit or receive data from said port. Therefore, the simpler approach made more sense.

To position the sensor correctly, we made sure that the sides of the sensor were not obstructed. This allowed for a proper air flow around the sensor.

4.2.1.2 *Requesting sensor data*

To get data from the sensor, a couple of actions have to be performed. First and foremost, the sensor has to be connected as described in the previous section. Once this is done, we can communicate using `/dev/ttyAMA0`. The framework that was supplied to us by the RUG DS people has a module providing such functionality, however, we chose not to use it for a couple of reasons:

- It uses a library which did not detect `/dev/ttyAMA0` to be a serial port. This caused us to not be able to communicate with the device.
- Input data was not limited to a size. This results in us receiving an arbitrary amount of bytes every time, and having to come up with some form of caching the data ourselves. Since we were working in a programming language unknown to us initially, it would have been too simple to make a mistake.

Please note that there was an ugly workaround for the first issue, being symlinking `/dev/ttyAMA0` to, for example, `/dev/ttyS80`, but since there were better alternatives, we chose to simply not use this module.

The alternative we chose initially was to use the `jSerialComm` library [12]. This library allowed us to hook into the underlying Java `InputStream` and `OutputStream`, making it possible to check if the amount of bytes required for a valid result were available. Using this library, the first beta version was presented to Sustainable Buildings.

In the end, however, we chose to convert the code to use the `Pi4J` library, simply because all our other sensors were implemented using that library.

Now that we have a library set up, we can communicate with the sensor. As defined in the documentation, there are three commands, namely the following:

- Calibrate zero point: This command calibrates the sensor to its zero point.
- Calibrate span point: This command calibrates the sensor to its span point.
- Read concentration: This command requests a read of the current concentration of CO_2 gasses (in ppm).

The first two commands are not particularly interesting for us. The last command is, as it gives us the data we need. The command to request data are the following nine bytes: `0xFF 0x01 0x86 0x00 0x00 0x00 0x00 0x00 0x79`.
The first byte is the starting byte, the second byte is the sensor number (which is, interestingly enough, a constant), the third byte is the actual command and the last byte is the checksum of the command (more on this later).

4.2.1.3 *Retrieving sensor data*

After having sent the `Read concentration` command, we get nine bytes back in the following format: `0xFF 0x86 0x?? 0x?? 0x47 0x00 0x00 0x00 0x??`. You'll recognise the starting byte and command byte from the request; they are identical. The third and fourth byte contain our data. It is not documented what the fifth byte does, but the example script by Dexter Industries (see Appendix A.1) uses it to calculate the temperature. As we have a separate sensor for exactly that, we chose to ignore it. Finally, the last byte is the checksum of the data.

The gas concentration, in particles per million, is calculated by multiplying the third byte (the high level concentration byte) by 256 and adding the fourth byte (the low level concentration byte). This technique allows for values up to 65535 in the output, which is plenty given that the sensor itself is designed to sense up to 2000 ppm [10]. Please note that you should not use a byte for the resulting value, as it will be well out of range. Instead, we used integers, although a char would have been enough.

To check if the data was transmitted properly, every command and return value has a checksum. We can calculate the checksum using the following formula:

$$0xFF - (byte[1] + byte[2] + \cdots + byte[7]) + 1 \tag{1}$$

If this value matches the value in the last byte, the checksum is correct and we can assume that no data corruption took place.

4.2.1.4 *Interpreting the sensor data*

After having retrieved some sensor data, we immediately send it to RabbitMQ for storage. We researched what kind of values are considered "healthy", and what values are absolutely not. It has been found that values of around 1400 ppm are usually found in office-like environments [13], and that decreasing this value to 945 ppm increases cognitive scores by 61% compared to regular office-like environments. Further decreasing the amount of CO_2 in the air to 550 ppm increases cognitive scores by 101% compared to regular office-like

environments. Any value under 1000 ppm adheres to the ASHRAE standards [14]. Given this information, we consider any values under 1000 ppm to be ”healthy”, and any values above 2000 to be unhealthy[15]. Between 1000 and 2000 are considered uncomfortable, as they may cause drowsiness due to poor air quality, but the air quality is still better than regular office-like environments. To convey this to the end-user, any value below 1000 is green, between 1000 and 2000 is orange, and every value above 2000 is red.

4.2.1.5 *Confirming correctly functioning of sensor*

At the start of the project we were dealing with a malfunctioning GrovePi board, which would result in incorrect values (ranging from incorrect checksums 99% of the time to CO_2 values of over 10,000). To confirm that the issue was fixed, we used a Netatmo weather station (see 2.2). As described there, this weather station was placed right next to the computer, so the CO_2 values were much higher than the values from the Healthy Pi, which was placed in a television stand near the ground. To test if our sensor was working correctly, we simply checked the values after the last person had left the room for a substantial amount of time. We noticed that this, indeed, resulted in extremely similar CO_2 readings, leading us to assume that our CO_2 sensor had been set up correctly.

This concludes the work on the CO_2 sensor.

4.2.2 *DHT*

For this project we used the DHT11 temperature and humidity sensor by Seeedstudio [16]. Its accuracy lies within 5% relative humidity and 2 degrees Celsius.

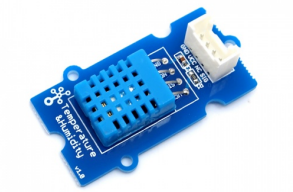

Figure 7.: DHT sensor

4.2.2.1 *Setting up the sensor*

It uses one of the digital ports of the Raspberry Pi, which means that we can connect to it and read data from it by using the I2C protocol.

The positioning of the sensor is not very important. We only made sure the sensor was not obstructed by objects such as walls or tables.

4.2.2.2 *Requesting sensor data*

As discussed in section 4.1, digital sensors like the DHT sensor sensor are reachable via the I2C protocol. This means that we can send a packet of four bytes to the sensor to connect with it.

By default, we have to send the following packet: `0x01, grovePin, 0x00, 0x00`, where 1 is the ”mode” for writing, and `grovePin` corresponds with the pin on which the device was connected (e.g. pin A0 is `i2cAddress` 0). However, one of the peculiarities of the DHT sensor is that you have to send a different command to read data; instead of sending $0x01$ we had to sent $0x28$. In addition to this, there are different versions of the sensor

available. One has to send the model of the sensor which is connected to the GrovePi (there are currently 4). The following models are available:

model	model no.
DHT11	`0x00`
DHT22	`0x01`
DHT21	`0x02`
DHT2301	`0x03`

Table 2.: Model numbers of the DHT sensor

Hence, the packet we have to send to the DHT sensor has become: `1, grovePin, 0x00, 0x00`.

4.2.2.3 *Retrieving sensor data*

The sensor returns a packet of nine bytes, of which index 1 to 4 contain the temperature and index 5 to 9 contain the relative humidity. This is due to the fact that the DHT sensor returns two floating point numbers, which take up four bytes each.
The temperature is in Celcius (° C) while the humidity is a percentage for the relative humidity (RH %).

4.2.2.4 *Interpreting the sensor data*

The temperature and relative humidity are interrelated. This means that the desired humidity depends on the (experienced) temperature. Based on the recommendations by the ASHRAE society [17], we found the following values:

season	relative humidity (RH %)	temperature (° C)
summer	30	24.5 - 28
	60	23 - 25.5
winter	30	20.5 - 25.5
	60	20 - 24

Table 3.: Desired temperature and humidity values

Humidity is also related to the perception of warmth. Besides that, extremely low humidity values (below 20-30%) are related to nose, skin, throat and eye irritation. On the other hand, high values pose problems for asthma patients, as well as a possible heat stroke, when a high humidity is combined with a high temperature (due to reduced evaporation from the body). [18]

4.2.3 *Light*

For this project we used the light sensor by Seeedstudio [19]. It has a GL5528 light detecting resistor (LDR) or photoresistor on board.

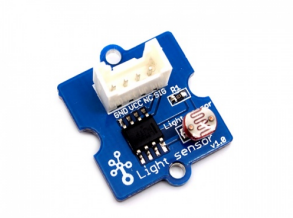

Figure 8.: Light sensor

4.2.3.1 *Setting up the sensor*

The sensor returns an analog signal to the GrovePi and should therefore be connected to one of the analog ports (`A0-A2`). Once connected, we may read data from the sensor using the `I2C` protocol. As for the placement, we suggest to place the sensor out of direct light rays. This way, the sensor will return an 'average' value rather than returning peak values.

To position the light sensor correctly, we had to make sure the sensor was not pointed at the wall or pointed at a window, as this would likely influence the data we get from the sensors.

4.2.3.2 *Requesting sensor data*

As discussed in section 4.1, analog sensors like the light sensor are reachable via the I2C protocol. This means that we can send a standard packet of four bytes to the sensor: `0x01, grovePin, 0x00, 0x00`, where 1 is the "mode" for writing, and `grovePin` corresponds with the pin on which the device was connected (e.g. pin A0 is `i2cAddress` 0).

4.2.3.3 *Retrieving sensor data*

The sensor returns a packet of four bytes, of which index 1 and 2 contain the value of the light sensor. This value by itself has no meaning, however two units may be derived from it: the voltage on the circuit as well as the resistance.

VOLTAGE To calculate the voltage, the following calculation should be applied:

$$v * ADC/1023 \tag{2}$$

Where ADC is a constant (analog to digital converter) that resembles the voltage of the sensor. By multiplying the analog value with the ADC value, and then dividing it by the range of the analog scale, we can calculate the current voltage on the sensor. Since the LDR is running on 5V, $ADC = 5$.

RESISTANCE To calculate the resistance, the following calculation should be used:

$$(1023 - v) * 10/v \tag{3}$$

We found out [20] that there exists an approximate formula for deriving the amount of Lux detected based on the characteristics of the sensor, and the value that it is sending through.

$$\gamma = \frac{\log(R_A/R_B)}{\log(E_B/E_A)} \tag{4}$$

Where R is the resistance, and E is the amount of light emitted. The values belonging to A are known from the sensor datasheet, as we will discuss next.

As can be seen from the graph and equation, the relation is influenced by a factor γ (ascend of the line), which is 0.6 for our sensor [21]. The coordinates on the line are a combination of the resistance and the amount of light emitted. Since the range of resistance of the sensor is known for 10 lx of light, which is 10 - 20 kΩ, we also have one part of the equation filled in. Because a range is given, we chose $R_A = 15$ kΩ (the average).

$$\log(E_B) = \frac{\log(R_A/R_B)}{\gamma} + \log(E_A) \tag{5}$$

$$\begin{aligned}
E_B &= 10^{\frac{\log(R_A/R_B)}{\gamma}+\log(E_A)} \\
E &= 10^{\frac{\log(15/R)}{0.6}+\log(10)} \\
&= 10^{\log(15/R)*10/6+1} \\
&= 10^{\log(15)*10/6-\log(R)*10/6} * 10 \\
&= (10^{\log(15)*10/6}/10^{\log(R)*10/6}) * 10 \\
&= (15^{\frac{10}{6}}/R^{\frac{10}{6}}) * 10 \\
&= 15^{\frac{10}{6}} * 10/R^{\frac{10}{6}} \\
&= 912.33/R^{\frac{10}{6}}
\end{aligned}$$

Hence, if the resistance R is known, the light emission E lx can be calculated.

4.2.3.4 *Interpreting the sensor data*

The recommended lighting levels vary given the use of the room. However, since the scope of the project is mainly limited to schools and office-like environments, we will shortly discuss their levels. We will also list some of the most common public spaces in buildings as a reference.

general office	300-500-750
computer work area	300-500-700
classroom / library	200 - 300 - 500
seminar room / art room	300 - 500 - 750
hallways	150 - 200 - 300
corridors / stairs / bathrooms	50 - 100 - 150

Table 4.: Lighting levels as researched by Indian Standard in lx [22]

As can be noted from the above examples, the average lighting levels needed for office-like environments vary around 400 lx for 'regular' typed offices, and rise to 750 lx for jobs that involve more concentration. On the other hand, rooms that do not need the additional lighting, typically have lighting levels of around 100 lx.

4.2.4 *PIR*

The sensor we use for our project to determine whether or not someone is in the room where the sensor is deployed is the Seeed-studio Grove PIR Motion sensor [23]. The creators of this sensor published, for both the sensor as well as the chip, the datasheets containing constraints and usages.

Figure 9.: PIR sensor

4.2.4.1 *Setting up the sensor*

The sensor is to be connected to a digital port on the GrovePi. Once plugged in, we can talk with the sensor using the I2C protocol.

As for the placement of the sensor; we had to make sure it was within six meters of our workstation, with a maximal angle of 120 degrees, as specified on the wiki of the sensor. Once this was done, the sensor was set up.

4.2.4.2 *Requesting sensor data*

To get data from the sensor, we had to use the I2C protocol, which is a very simple and easy-to-use protocol. In our particular setup, the I2C bus was located at `/dev/i2c-1`. Using the `pi4j` library [24], we were able to send commands to the sensor with two lines of code; The first initiates the device object using the `i2cBusName` and `i2cAddress` (which, on our setup, was always four), and the second line sends the following command on the bus: `0x01, grovePin, 0x00, 0x00`, where 1 is the ”mode” for writing, and `grovePin` corresponds with the pin on which the device was connected (e.g. pin D8 is `i2cAddress` 8).

4.2.4.3 *Retrieving sensor data*

Retrieving the data is extremely simple: we simply read a byte. The only two possible output values are 0 and 1, with 0 being ”no motion detected” and, conversely, 1 being ”motion detected”.

4.2.4.4 *Using the sensor data*

Currently, the sensor data is not being used. The sensor data is designed to be used for actuators. A great example is the temperature, it makes no sense to start heating or cooling a room if there is no-one in it. In a future stage, the data may be analysed to allow for predicting when the first person will enter a room, as to make pre-heating a room possible, but this is definitely out of scope for our project.

5

DEPLOYMENT

To do our analysis, Sustainable Buildings [2] deployed several sensors for us in schools in Groningen. In this chapter, we will list the schools and explain where and how the sensors were placed.

5.1 GRONINGSE SCHOOLVERENIGING

The first three sensors were placed in a building of the Groningse Schoolvereniging (GSV, *Groningen School Association*) [1], which is located in the south of the city of Groningen.

In each of the three deployment rooms there is an air ventilation system in place, taking care of both the supply as well as draining of air.

Figure 10.: Deployment of sensors in the GSV school

1 http://g-s-v.nl/

5.1.1 *Room 1 and 2*

These rooms are controlled by two different building management system (BMS) developed by Kieback & Peter [2]. Their BMSes measure CO_2 levels, temperature and humidity. The first sensor is placed adjacent to a window, whilst the second is placed in the corner of the room.

5.1.2 *Room 3*

This room is in a separate building from rooms 1 and 2 and is also controlled by a different BMS. This system is developed by Priva [3]. This sensor is placed on a wall, just below the ceiling.

5.2 DE REGENBOOG

The latter two sensors were deployed in the SKSG Regenboog school [4], which is located in the north of Groningen.

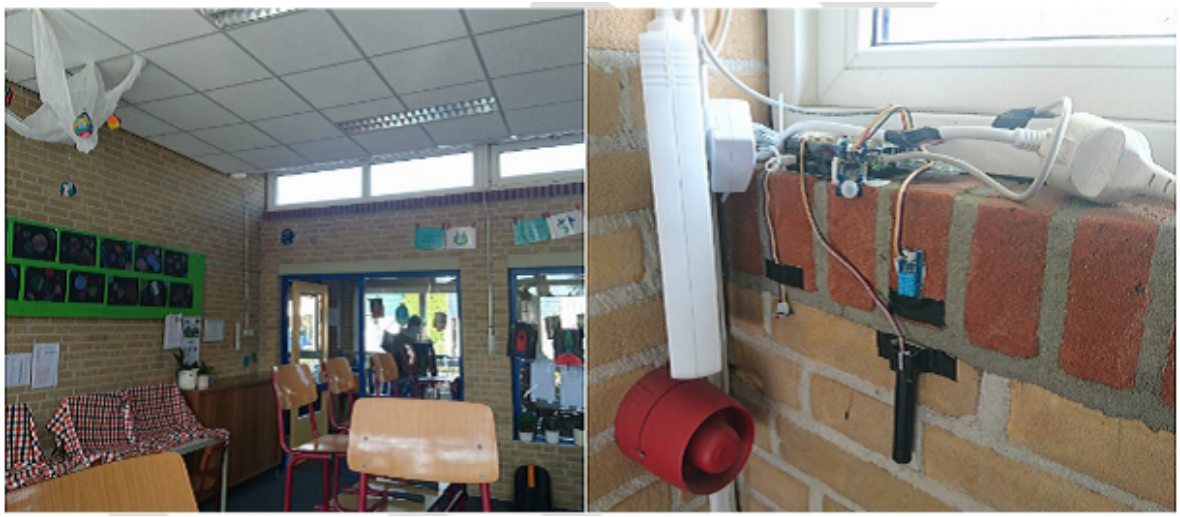

Figure 11.: Deployment of sensors in the Regenboog school

5.2.1 *Room 1 and 2*

In both classrooms there is air ventilation which relies on the natural movement of air (caused by wind and temperature differences). The ventilation is made possible by windows which can be opened in the frontage. There is no BMS taking care of the ventilation or heating.

Both sensors are placed just below the ceiling, next to a window which may be manually opened for ventilation purposes.

2 http://www.kieback-peter.de/de-en/products/building-management-system
3 http://www.priva.co.uk/
4 http://sksg.nl/locaties/sksg-regenboog/

Part III

RESULTS

6

SETUP

A large part of the Healthy Pi project was building the Healthy Pi. Using a Raspberry Pi and a GrovePi+, we were able to build a platform that is both fairly mobile, easy to use and provides support to a large array of sensors out-of-the-box. Having chosen for four different types of sensors (see chapter 4), we were able to retrieve data relevant to the healthiness of a room. By sending this data to the RabbitMQ server, it got stored in the Cassandra database, after which we could request the results via the REST server. Given the researched ranges of what kind of value from a sensor is considered healthy, we were able to reasonably see how healthy the room in which the Healthy Pi was deployed is.

Figure 12.: The Healthy Pi

7

LITERATURE

Using the data gathered by the Healthy Pi, we are able to assess if a room is healthy or not. Using information from articles, as detailed in the section of the relevant sensor, we have come to an understanding what makes a room healthy. The values are the following:

Value type	Value unit	Desired	Uncomfortable	Unhealthy / Counterproductive
CO_2	PPM	<1000	1000-2000	>2000
Light	Lux	500	<300, >750	
Temperature [1]	Celsius	20-25.5	<20, >25.5	
Temperature [2]	Celsius	23-28	<23, >28	
Humidity	Relative (%)	30-60	20-30, 60-80	<20, >80

Table 5.: Sensory values related to their healthiness

NOTE Excessive amounts of light (more than 750) may be desired in some cases. For these specific examples, please see section 4.2.3.

NOTE The desired temperature in the above table are extremes; the actual temperatures also depend on the humidity. For a more elaborated table, see section 4.2.2.

1 This row contains values for the winter season.
2 This row contains values for the summer season.

8

EXPERIMENTAL RESULTS

Using the data from the sensors, a dashboard was created to show the general state of a room at the current moment. It uses the latest data it can find and polls for the latest data in the background every so often, as to keep the dashboard up to date with the environmental changes in the room. A screenshot of what this dashboard looks like:

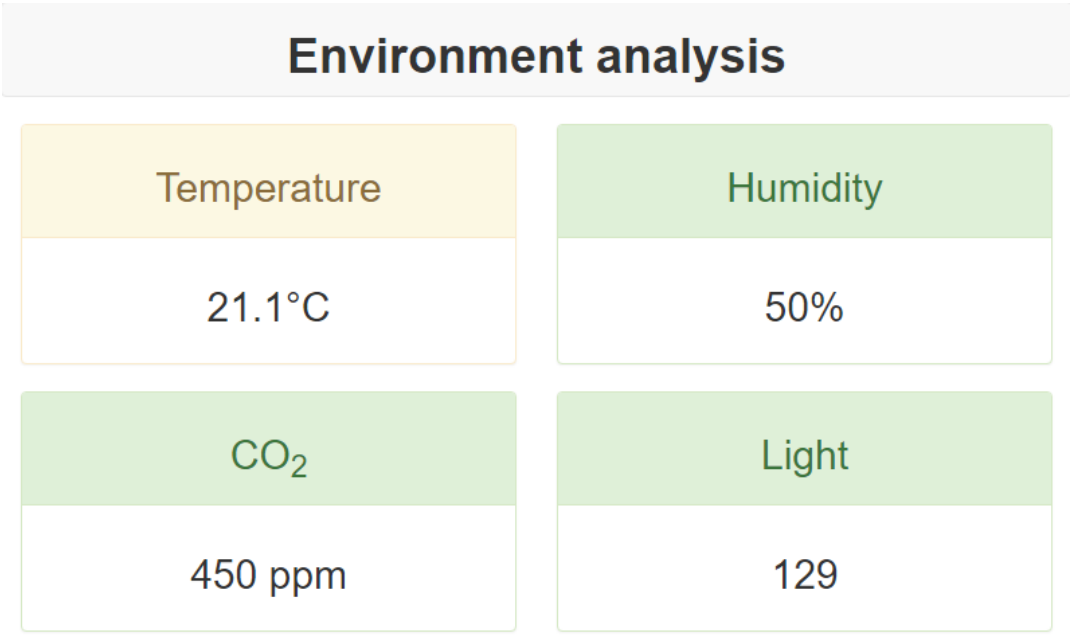

Figure 13.: Website displaying status of room

As you can see, the temperature of the room is not quite in the range of what is deemed 'comfortable', but is not worrisome. Furthermore, the values for the humidity, CO_2 and light are all fine. Please note that this website is still in alpha and may undergo massive overhauls in the coming time.

Please also note that the dashboard is not public and may never be public. It was mainly created as a proof of concept and to view the healthiness of a room without having to query each sensor manually.

8.1 SENSORS

In the following sections we will try to analyze the plots of section 8.3. There are also some general remarks we have with regards to these plots.

First of all, there are some data points missing in the plots because of a database error. This can be clearly concluded from the straight lines on the temperature and humidity plot (figure 16), as well as the light value plot (figure 18). The database problem occurred between June 23rd and June 24th.

Furthermore, the light sensor sometimes shows values under zero. These values were caused by a bug. The bug has been fixed, but the fix has not yet been applied.

8.1.1 CO_2

As can be seen from figure 14, there are usually five spikes followed by a gap of about two days. This clearly indicates the weekly cycle, with of course the spikes representing the weekdays. When we zoom in on such a week, we notice a few things: during weekdays, the CO_2 levels rise at around 8:00 AM. The levels will then fluctuate between 700-800 PPM and 500 PPM, except for the Regenboog sensors, whose levels are slightly higher on average.

This is the case until around 3:00 PM, the moment at which the levels will gradually decrease over time. This decrease can be explained from the fact that the sensors are deployed in schools; classes usually end at around 3:00 PM in Holland. However, on Wednesdays the classes usually end slightly earlier, which is also displayed in the graph of the CO_2 levels.

Another remarkable thing that we mentioned above, is that the levels for Regenboog sensors are rather high. A possible reason for this could be that the rooms in which the sensor have been placed is relatively cramped, as opposed to the other rooms. On the other hand, there may also be a serious ventilation problem, or there may simply be a lot of children occupying them.

setup name	# data points	min	max	mean
GSV-C2	5236	213	1100	498.4
GSV-C3	4844	355	792	454.0
RegenboogC1	3433	213	1772	538.8
RegenboogC2	2175	292	2000	557.6

Table 6.: Analysis of CO_2 value data points

8.1.2 *Temperature and humidity*

The temperature and humidity plot in figure 16 shows that the humidity and temperature are both rather constant values. There are only minor fluctuations to be spotted in the graph.

We did notice that both the temperature as well as the humidity values are in the range that we defined as acceptable in section 4.2.2: the temperature fluctuates around 22 degrees Celsius and the humidity lies perfectly within the 40-60% boundary.

A remarkable thing that is visible in figure 17 is that the humidity levels in each of the GSV rooms are nearly identical, except for a small offset. Even the gap (because of a database problem) is the same.

The same can be concluded from the temperature, which is also rather equal in each of the five rooms. However, these levels lie closer to each other than the humidity levels, and while the humidity in the Regenboog differs from the GSV rooms, their temperature shows more similarities. This is possibly caused by natural warmth, combined with room activity - there seems to be no heat regulation.

Besides this, during our measurements there was a short heat wave in Holland. This heat wave occurred between June 21 and 24, which is also noticeable from the graph: both the temperature and the humidity increased slightly (not to be confused with the database error).

setup name	# data points	T min	T max	T mean	H min	H max	H mean
GSV-C1	2950	19	29	21.7	37	63	43.9
GSV-C2	3488	20	25	21.8	43	80	54.5
GSV-C3	3025	18	21	19.1	34	79	46.7
RegenboogC1	2156	19	27	22.4	39	59	44.5
RegenboogC2	1430	21	29	25.0	42	58	46.9

Table 7.: Analysis of DHT value data points

8.1.3 *Light*

The overall plot in figure 18 clearly shows a repetition of a daily pattern, in which there is no distinction possible between weekdays and weekend. It also shows that the three sensors that are plotted are each detecting an equal amount of light.

When we zoom in on a weekly level in figure 19, we notice a similar fashion in the values. However, in the weekend (June 18th and 19th), the amount of lighting shows signs of fluctuation much more often than during weekdays. This may be caused by the fact that during the weekend, the class rooms are not occupied and hence, lighting is switched off. During a normal weekday, lighting will make sure that there is enough light, when there is not enough sunlight shining into the room already. Since the lighting is switched off in the weekend, the amount of lighting will be much more dependent on the amount of sunlight.

NOTE The values plotted are not lux values. The values that the sensor returns could not be converted to lux using the algorithm discussed in section 4.2.3. However, given the scale of the sensor (which is 0-1023), it is possible to detect relative highs and lows.

NOTE There are values below zero in the plot. This is the result of a bug in the code, which was discovered in a late stage of the project. These values are not possible; please ignore them.

setup name	# data points	min	max	mean
GSV-C2	3485	0	639	282.6
GSV-C3	3220	0	612	133.2
RegenboogC1	2288	0	639	252.9
RegenboogC2	1451	0	639	275.1

Table 8.: Analysis of light value data points

8.2 CONCLUSION

Using sensory data, one can determine the healthiness of an office space by checking if the sensed values are within the acceptable ranges for sources that directly impact health (such as CO_2), or are in a comfortable range for sources that affect performance (such as light, temperature, humidity). These ranges have been specified in 5.

As for the CO_2 sensor, the values in the Regenboog rooms are slightly higher and are thus considered to be less healthy. Both the temperature and as well as humidity appear to be varying a lot, but they are not within the unacceptable ranges.

Due to a problem with the conversion of the light values, we cannot conclude anything about the healthiness or comfort of a room based on the light levels. For a reason unknown, the values that the sensor returns can be converted to voltage and resistance, but these do not show the relation as described in section 4.2.3.

Hence, we may conclude that each of the rooms is equally healthy, with the exception of the Regenboog rooms, which show irregular CO_2 spikes. The spikes however, should not pose any problems to health as they are still in the acceptable range.

8.3 PLOTS

The following pages contain plots of the CO_2, DHT and light sensor data, respectively.

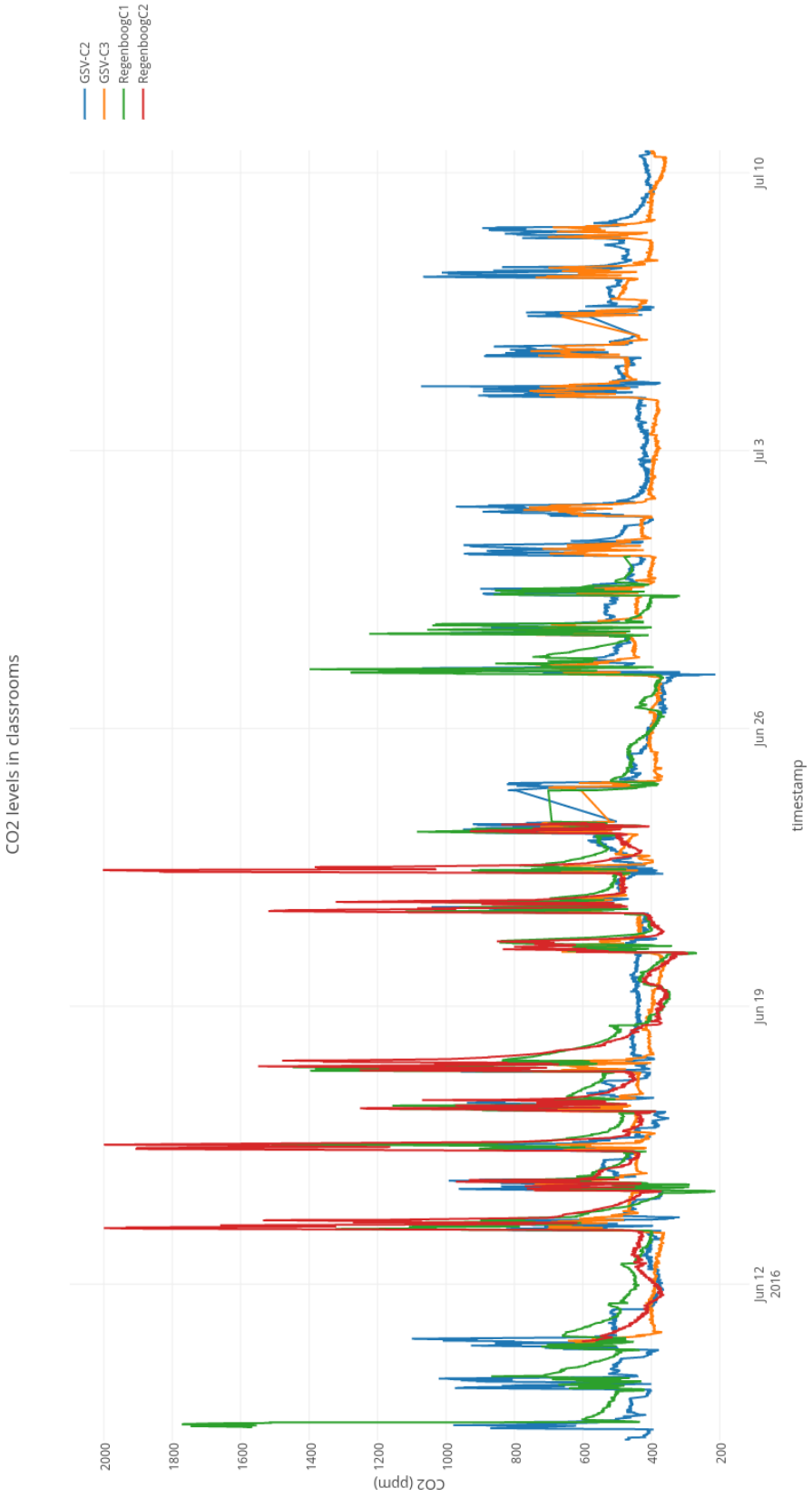

Figure 14.: Overall CO_2 value plot

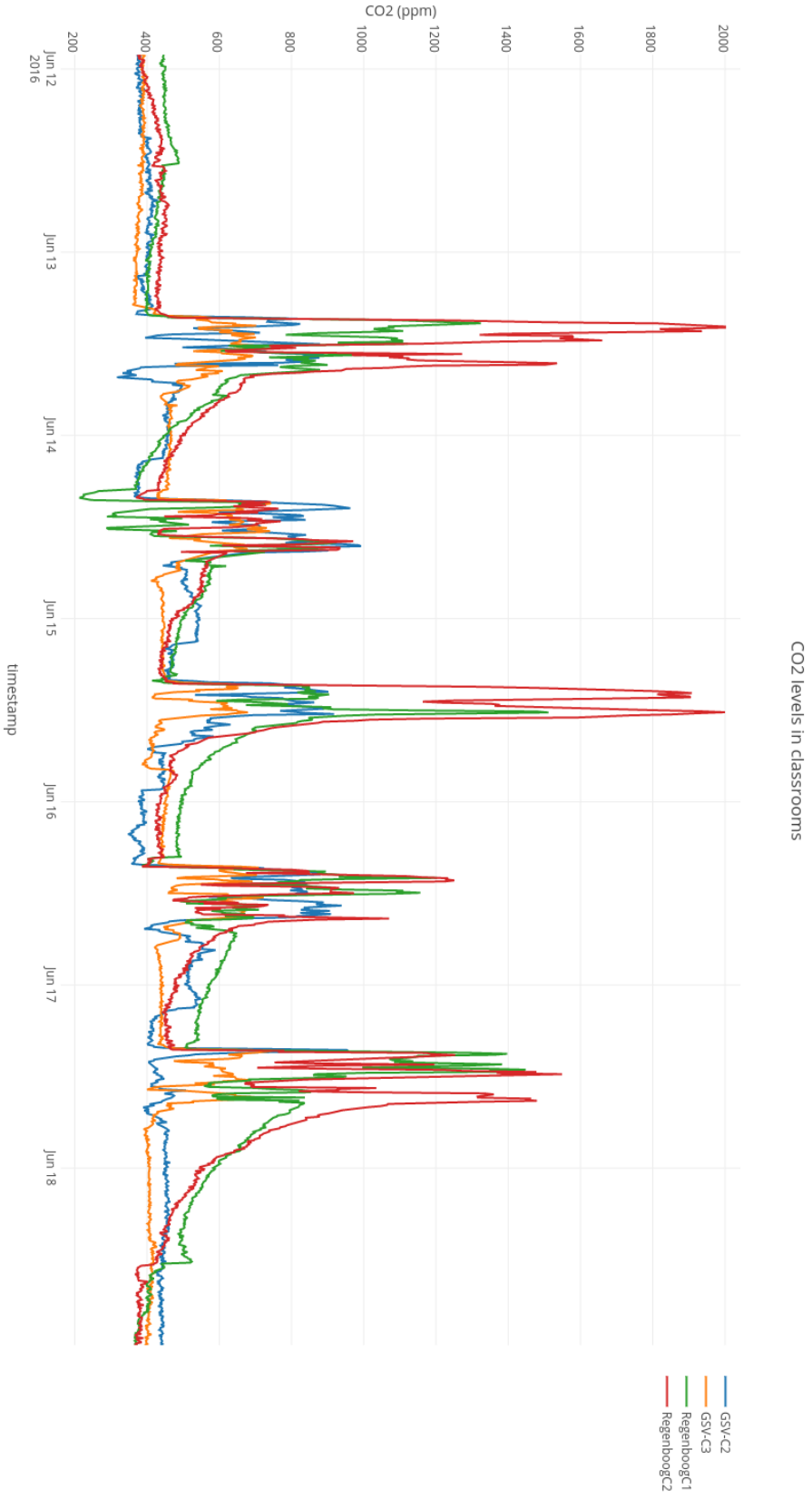

Figure 15.: Weekly CO_2 value plot

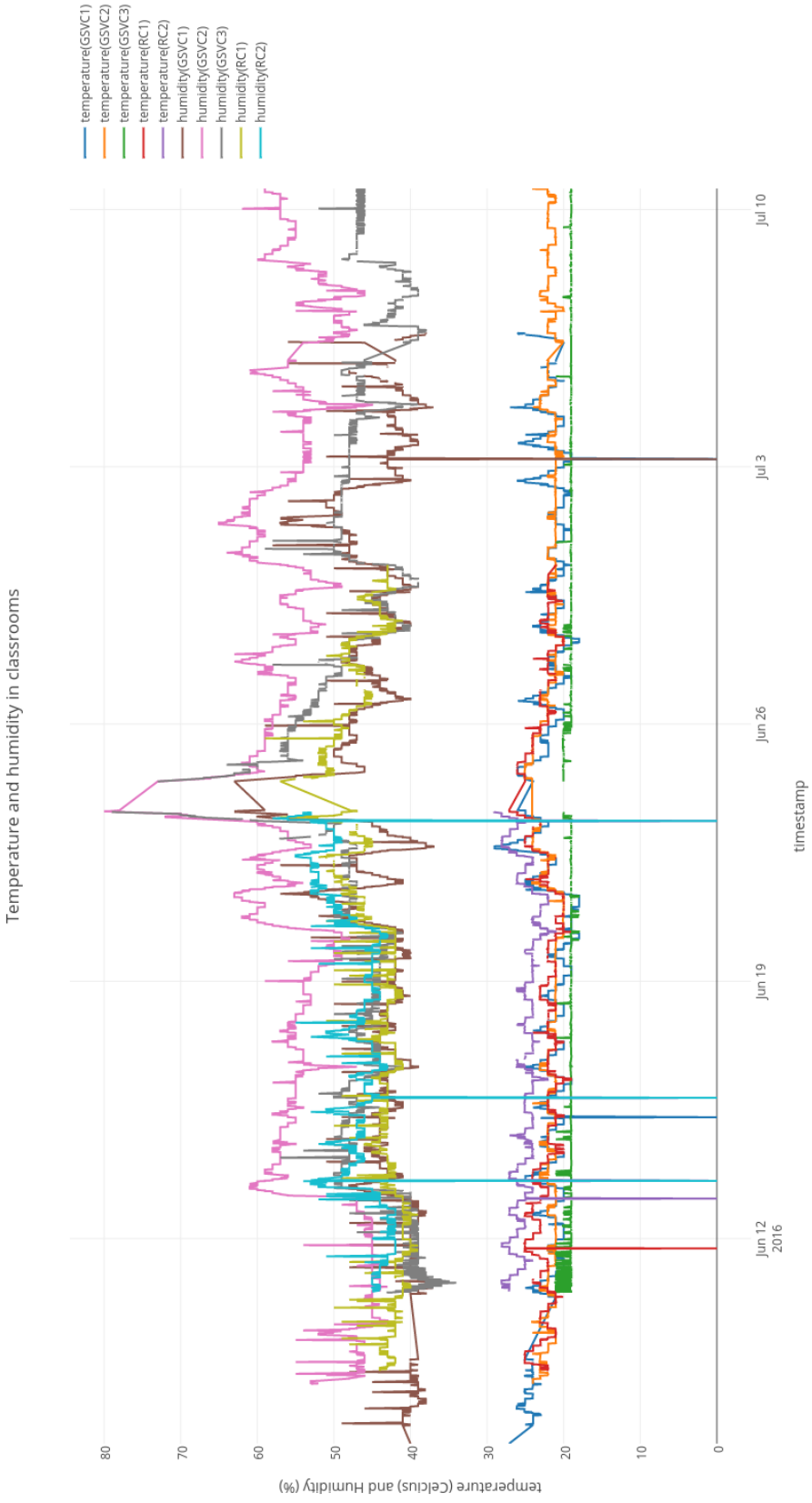

Figure 16.: Overall temperature and humidity plot

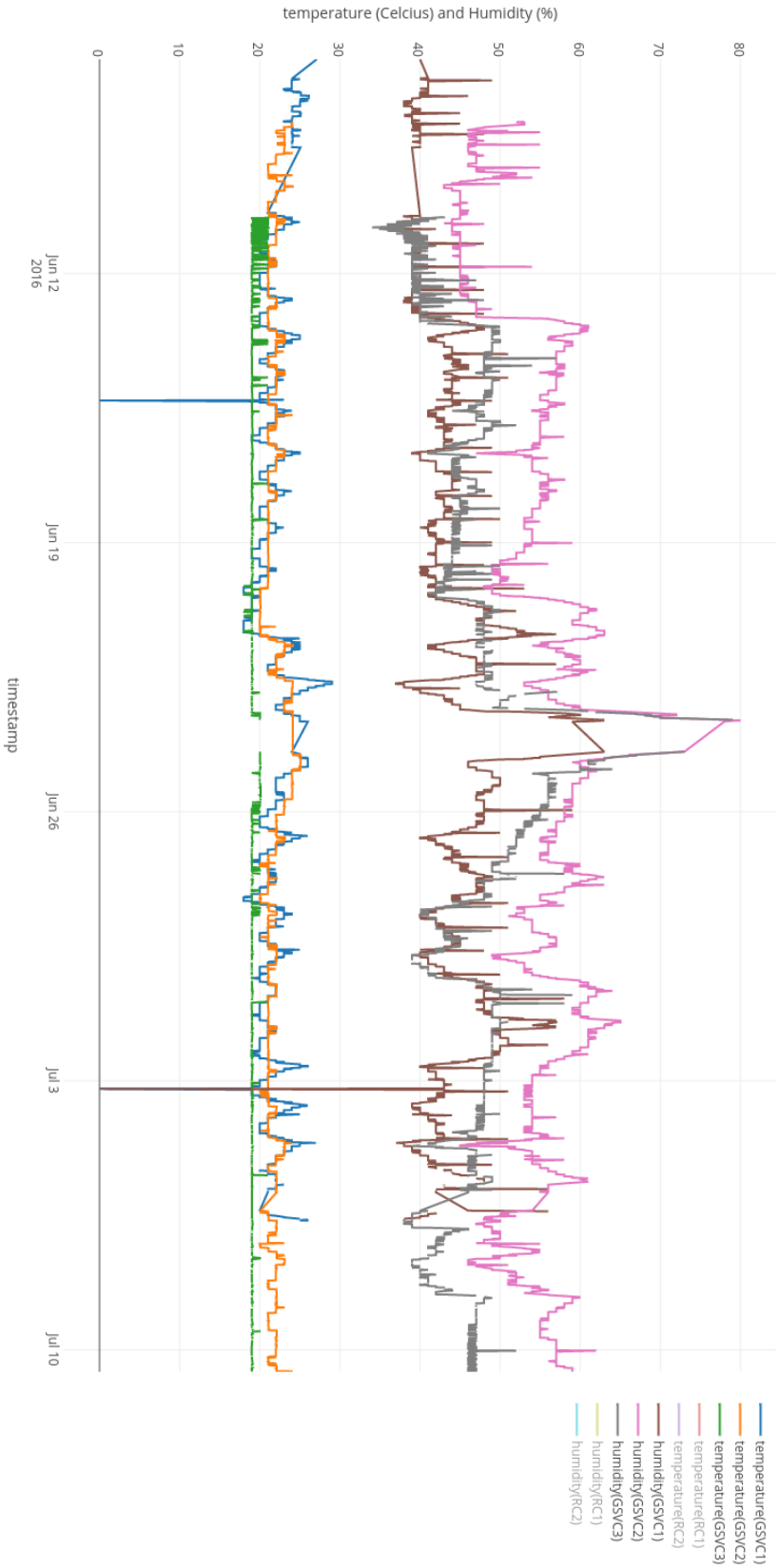

Figure 17.: Comparison temperature and humidity plot (K&B vs Priva)

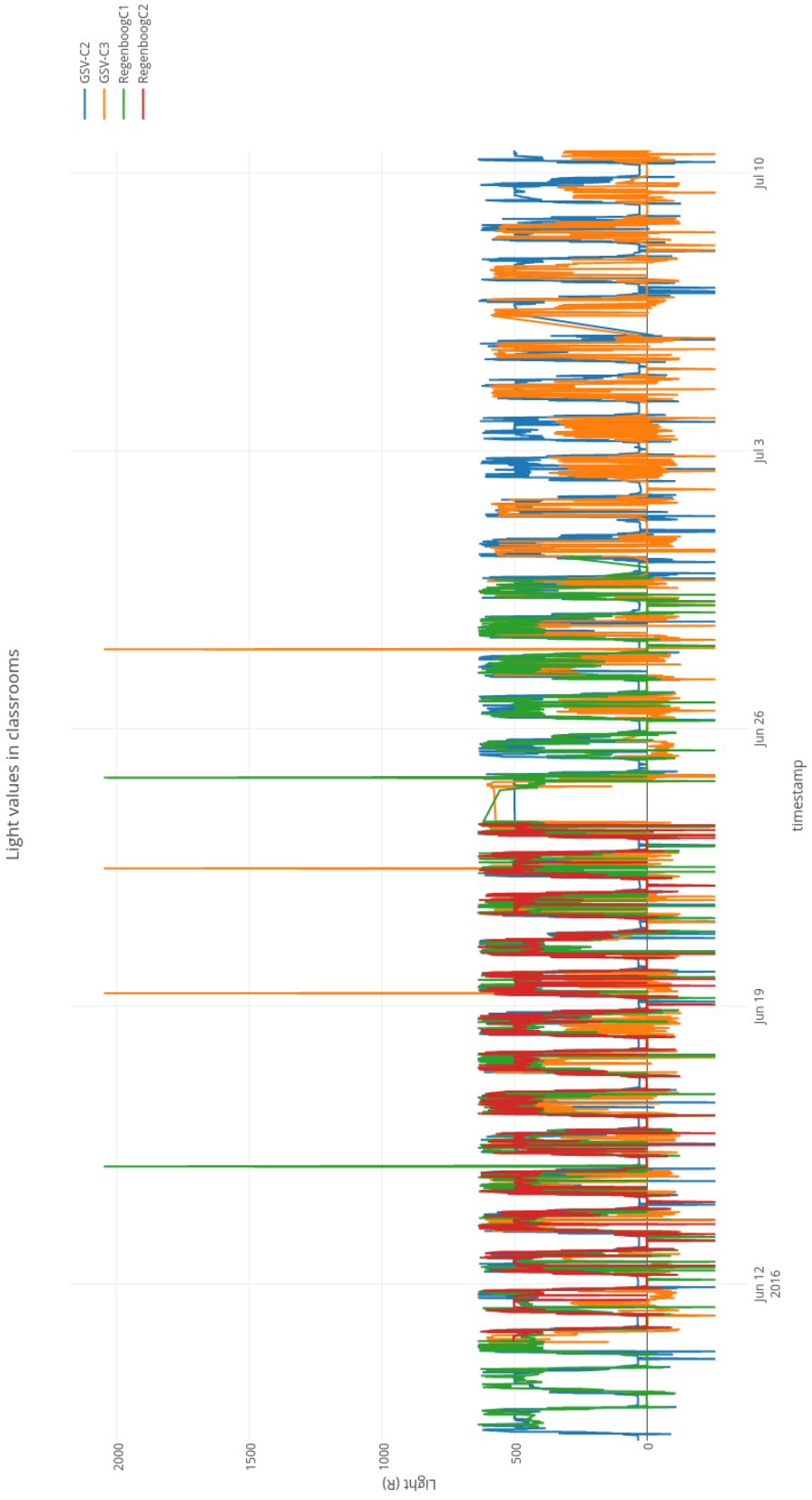

Figure 18.: Overall light value plot

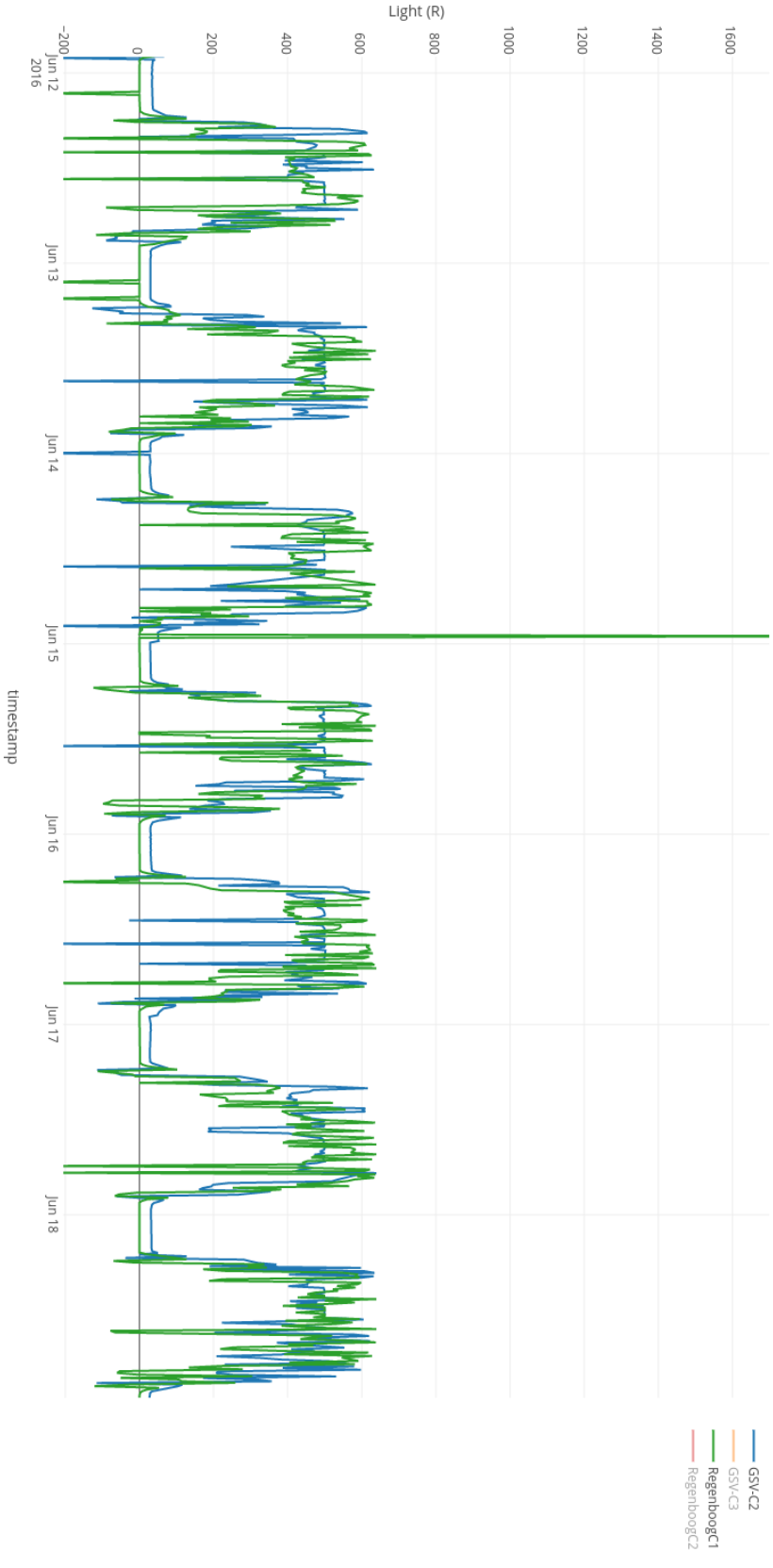

Figure 19.: Comparison light value plot (GSV vs Regenboog)

Part IV

CLOSURE

9

CONCLUSIONS

We were presented with a problem: How does one determine the healthiness of an office space based on sensory data? In order to be able to solve this problem, we subdivided the problem into several sub-problems:

Is it possible to determine the condition of an office space with a small set of sensors, namely CO_2, temperature, humidity and light sensors? (2)

How should the aforementioned sensors be used? (3)

How do the values of the sensors relate to the healthiness of a room? (4)

Is the collected data sufficiently significant and comparable? (5)

Clearly, the answer to our first question is positive. We have developed the Healthy Pi using a Raspberry Pi, a GrovePi and a small set of sensors (as stated in the question), deployed this device to a number of sample locations and gathered information from the sensors about those locations. To fully answer this question, however, we must first answer questions 3 and 4.

Everything needed to answer question 3 has been answered in chapters 3 through 5, with the general conclusion being that we can use the aforementioned sensors to our advantage by plugging them into a GrovePi, which is in turn on top of a Raspberry Pi, and putting them in a room. The manner in which we use the sensors is simple; we deploy them and gather data from the sensors using code we wrote. If needed, we convert the data, and finally push the data into our database for analysis.

Using the data gathered and research conducted, as detailed in chapters 7 and 8, we can now answer question 4. Our literature study concluded that some aspects of an environment may not directly influence a person his health. It may cause discomfort, which may lead to a decrease in working efficiency for whoever is in that room, but it does not affect the healthiness of the room. For example, a surgeon would appreciate a lot of light in an operating room, so he can see everything clearly, while a general office does not need such excessive amounts of light.

Similarly, the humidity and temperature influence the comfort of a working place rather than the healthiness. Laboratories usually have strict environmental control, but the increased (or decreased) temperature and/or humidity does not affect its researchers health.

CO_2 is a different story, however. In general, one might say that the higher the level of CO_2 in the air, the less healthy a room becomes. In section 4.2.1 it is detailed how the CO_2 sensor works and what values are OK. Generally speaking, a room with less than 1000 CO_2 ppm is considered "healthy", while any concentration between 1000-2000 ppm is considered uncomfortable and may cause drowsiness due to poor air quality. Above the concentration of 2000 CO_2 ppm it becomes "unhealthy", which usually results in headaches, loss of concentration, etc [15]. For a nice overview of what values are good or bad, see table 5 in chapter 7.

Now that we have answered questions 3 and 4, we can come back to question 2. Using the data from the sensors and the information gathered through literature studies, we have come to the conclusion that it is indeed possible to determine the condition of an office space with a small set of sensors. We have elaborated on the sensors on what sensors we use, how they work, what data we expect to get and what this data means. Using this data, we can find out whether they adhere to healthy or comfortable values. This data may be requested by users through a dashboard, which shows a nice overview of the current state of their room.

Finally, we are also able to answer question 5. We have compared the data of our sensors to the data coming from a device that is being sold to the public, the Netatmo [8]. The data that we compared (CO_2, temperature and humidity, the Netatmo does not measure light) is extremely similar. Furthermore, taking into account that no sensor is 100% accurate, we checked if the difference between the sets of sensors was within the documented accuracy, which was indeed the case. You can find the documented accuracy at the wiki page of each respective sensor, as listed in chapter 4.2.

Therefore, in short, the answer is: Yes, the results of our research are sufficiently significant and comparable. We have done sufficient research towards each of the sensors and what values are considered "healthy" or "comfortable". We have compared the values from our sensors to comparable technology and found them extremely similar.

Having given answers to all our sub-problems, our main research question 'How does one determine the healthiness of an office space based on sensory data?' is answered. We have looked at sensors, what data they produce, how we obtain this data, how reliable the sensors are and what this data means with regards to the healthiness of a room.

10

FUTURE WORK

While the Healthy Pi is a great piece of technology, it can definitely be extended to offer additional functionality to further increase the effectiveness of such a device in, for example, an office space. This chapter will outline some of the possible extension for the Healthy Pi and its related projects.

10.1 ACTUATORS

The Healthy Pi does not take any actions on its own. A user or moderator can view the values that the Healthy Pi is gathering and propose action(s) accordingly. This process can be mostly automated though. The Healthy Pi was designed with actuators in mind, which is why the motion sensor was added and implemented. Actuators can also be used to increase the energy efficiency of a room, by, for example, decreasing the intensity of the lighting when it is a sunny day, or turning off the heater on such a sunny day.

10.2 FIX LIGHT SENSOR

As described in section 8.1.3, there were problems converting the returned values to lux values. Using the equation as described in 4.2.3, we could not convert the values to lux. We were thus only able to compare the values with each other, and could not determine the healthiness or comfort of a room based on lux values. To make use of the data, one should implement the conversion (correctly).

10.3 ADDITIONAL SENSORS

The sensors we chose do not give a full picture of the healthiness of a room. Extra sensors such as a sound sensor or a dust sensor will provide additional information regarding the state of a room, whether it be improved occupancy detection or extra information about the environment. These extra sensors may also help in decision making, as ventilation should probably be enabled (or increased) if the amount of dust in a room is high, even though the CO_2 levels may be fine. On the other hand, sometimes there exist multiple solutions to the same problem: opening a window will have the same effect as an air condition system, while the latter solution will also have a negative effect on the humidity (which the first has not). Adding extra sensors will aid in making the best decision given all information/constraints.

10.4 RESEARCH ON ENERGY EFFICIENCY

Our research focuses primarily on the health of people in a room with a Healthy Pi, e.g. what are ”healthy” values and what values are not considered healthy. Additional research should go into how to make a room energy efficient, while simultaneously keeping the healthiness of a room preserved.

Part V

METHODOLOGY

11

CHALLENGES

Practically all (if not all) projects will face several problems during the development phase. So did we, and in this chapter we will try to list most of the important ones, including how we managed to overcome or resolve these issues.

11.1 BROKEN HARDWARE

In the first week of our project development we received our first set, consisting of a Raspberry Pi, a GrovePi+ and a set of sensors. We managed to get each of the sensors running and to get data out of them. However, the CO_2 sensor returned arbitrary values with no consistency at all. It would return five bytes sometimes, then 35, then eleven, etc.

Our supervisor initially gave us a new CO_2 sensor, to see if the sensor was simply broken. However, our issues persisted. It was clear the sensor was likely not the root of the problem.

Our supervisor then gave us a new GrovePi+. After giving it a spin, the CO_2 sensor gave far more reasonable values. The values of the other sensors remained the same. As we were using a Netatmo device providing us with the same kind of data, we could from that moment on verify our sensor data as well. This immediately resolved the issue.

11.2 UNKNOWN PROGRAMMING LANGUAGE

The development was off to a slow start due to us being inexperienced in the development language, Scala. Besides that, we had to get ourselves accustomed with the framework, which was also written in Scala. Because we did not anticipate this steep learning curve, we had a small delay in the first phase.

We tried to gain knowledge from the existing sources being available to us (from other parts of the RUG Distributed Systems framework). This helped us to get back on track and get our first sensor working. Given the similar setup for the rest of the sensors (except for the CO_2 sensor), we could quickly recover from the delay.

11.3 INCORRECTLY FUNCTIONING LIBRARY

For the GrovePi CO_2 sensor we made use of a different library at first, a library incorporated into the framework we were using. Using this library we immediately ran into an issue, since it could not find our sensor, while it was connected correctly and it was detected by the operating system as well. After several days of investigation and looking for answers, we found another user in a community who experienced the same issue. Apparently, the library was looking for a completely different device than it was supposed to. Using a symbolic link in the file system fixed the issue. Clearly this should not have been needed.

11.4 INCORRECTLY FUNCTIONING INTELLIJ BUILD

During the development phase of the project, IntelliJ released several updates for the software, as well as the writer of the Scala plug-in for IntelliJ. One of these updates caused the RUG Distributed Systems framework to display errors in the code (by syntax highlighting). Since this issue required a combination of several constraints to be met, it was at first hard to reproduce. We found out that the error was inside IntelliJ, since our compiler was still functioning correctly. Once we found this out that the error was on IntelliJ's side, as opposed to the developers of the RUG DS framework, a bug report was filed and the issue was fixed within days.

11.5 INCORRECT CODE

The GrovePi set comes with several scripts and projects which are standalone. This allows one to quickly set up a testing environment, test sensors and verify their output. In addition to this, there exist a lot of wiki pages which have been created by the developers and the community.

A small mistake in one of the sources caused one of the scripts to pull data from one of our other sensors, rather than the sensor it was intended to monitor. This mistake was not reported on the wiki, manuals, nor in the code itself (comments indicated otherwise). Once discovered, this led to a pull request from our side to the developers, eventually fixing the issue.

Part VI

APPENDICES

A

CO_2 EXAMPLE PYTHON CODE

Listing A.1: CO_2 example python code

```
1  import serial, time
2  import struct
3
4  ser = serial.Serial('/dev/ttyAMA0',  9600)
5
6  class CO2:
7      inp =[]
8      cmd_get_sensor = "\xff\x01\x86\x00\x00\x00\x00\x00\x79"
9
10     def __init__(self):
11         ser.flush()
12
13     def read(self):
14         try:
15             ser.write(self.cmd_get_sensor)
16             self.inp = ser.read(9)
17             high_level = struct.unpack('B',self.inp[2])[0]
18             low_level = struct.unpack('B',self.inp[3])[0]
19             temp_co2  =  struct.unpack('B',self.inp[4])[0] - 40
20             conc = high_level*256+low_level
21             return [conc,temp_co2]
22
23         except IOError:
24             return [-1,-1]
25
26 if __name__ == "__main__":
27     c = CO2()
28     while True:
29         print(c.read())
30         time.sleep(1)
```

Source: https://github.com/DexterInd/GrovePi/blob/master/Software/Python/grove_co2_sensor/grove_co2_lib.py

ACKNOWLEDGEMENTS

We would like to thank all the people at Sustainable Buildings for providing feedback on how to more forward when the sensors were against us. Your input was very valuable to us and definitely helped steer the project in the correct direction.

In particular, we would like to thank Brian Setz for always being available for questions regarding the framework or Scala in general. Without his help, we would still be wondering how Scala actually works. His help contributed greatly to the success of the project. The feedback he gave on our submitted code was very critical, and rightfully so. Our code quality increased immensely thanks to him.

Furthermore, we would like to thank Tuan Anh Nguyen for his continued effort into thinking with us on how to improve the project, and for supplying us with the required hardware to test our environments.

Lastly, we would like to thank Will Ceolin for developing the front-end of our application.

LIST OF FIGURES

LIST OF TABLES

LISTINGS

BIBLIOGRAPHY

[1] Office of Energy Efficiency & Renewable Energy. (2016). About the commercial buildings integration program. visited on 2016-06-27, [Online]. Available: http://energy.gov/eere/buildings/about-commercial-buildings-integration-program.

[2] RUG DS. (2016). Rug distributed systems website, [Online]. Available: http://www.cs.rug.nl/ds/.

[3] S. M. Joshi, "The sick building syndrome", *Indian Journal of Occupational and Environmental Medicine*, pp. 61–64, Dec. 2008. DOI: 10.4103/0019-5278.43262. [Online]. Available: https://www.epa.gov/sites/production/files/2014-08/documents/sick_building_factsheet.pdf.

[4] eranation. (2016). Coderwall — cake pattern in scala / self type annotations / explicitly typed self references - explained. visited on 2016-07-11, [Online]. Available: https://coderwall.com/p/t_rapw/cake-pattern-in-scala-self-type-annotations-explicitly-typed-self-references-explained.

[5] Seeedstudio. (2016). Seeedstudio website, [Online]. Available: http://seeedstudio.com/.

[6] D. Industries. (2016). Dexter Industries GrovePi sources, [Online]. Available: https://github.com/DexterInd/GrovePi.

[7] Seeedstudio. (2016). Category:grove - wiki. visited on 2016-07-09, [Online]. Available: http://seeedstudio.com/wiki/Category:Grove.

[8] Netatmo. (2016). Netatmo website, [Online]. Available: https://www.netatmo.com/.

[9] Archos. (2016). Archos weather station. visited on 2016-07-07, [Online]. Available: http://www.archos.com/gb/products/objects/chome/aws/index.html.

[10] Seeedstudio. (2016). Grove - CO2 Sensor. visited on 2016-06-21, [Online]. Available: http://www.seeedstudio.com/wiki/Grove_-_CO2_Sensor.

[11] Zhengzhou Winsen Electronics Technology CO., LTD. (2016). MH-Z16 Intelligent Infrared Gas Module Manual, [Online]. Available: http://www.seeedstudio.com/wiki/images/c/ca/MH-Z16_CO2_datasheet_EN.pdf.

[12] W. Hedgecock. (2016). jSerialComm library. visited on 2016-06-21, [Online]. Available: https://github.com/Fazecast/jSerialComm.

[13] J. G. Allen, P. MacNaughton, U. Satish, S. Santanam, J. Vallarino, and J. D. Spengler, "Associations of cognitive function scores with carbon dioxide, ventilation, and volatile organic compound exposures in office workers: A controlled exposure study of green and conventional office environments", *Environmental Health Perspectives*, vol. 124, no. 6, pp. 805–812, Jun. 2016.

[14] S. Petty. (1999). Ashrae's position on carbon dioxide, [Online]. Available: www.eesinc.cc/downloads/CO2positionpaper.pdf.

[15] Kane. (2016). What are safe levels of co and co2 in rooms? — kane international ltd. visited on 2016-07-12, [Online]. Available: https://www.kane.co.uk/knowledge-centre/what-are-safe-levels-of-co-and-co2-in-rooms.

[16] Seeedstudio. (2016). Grove - Temperature and Humidity Sensor. visited on 2016-06-21, [Online]. Available: http://www.seeedstudio.com/wiki/Grove_-_Temperature_and_Humidity_Sensor.

[17] American Society of Heating, Refrigerating, and Air Conditioning Engineers. (2010). Ashrea standard 55. visited on 2016-06-22, [Online]. Available: https://www.ccohs.ca/oshanswers/phys_agents/thermal_comfort.html.

[18] A. V. Arundel, E. M. Sterling, J. H. Biggin, and T. D. Sterling, "Indirect health effects of relative humidity in indoor environments", *Environmental Health Perspectives*, vol. 65, pp. 351–361, Mar. 1986.

[19] Seeedstudio. (2016). Grove - Light Sensor. visited on 2016-06-21, [Online]. Available: http://www.seeedstudio.com/wiki/Grove_-_Light_Sensor.

[20] RoboticLab. (2016). Photoresistor. visited on 2016-06-22, [Online]. Available: http://home.roboticlab.eu/en/examples/sensor/photoresistor.

[21] Senba Optical & Electronic CO., LTD. (2016). GL55 Series CdS Photoresistor Manual. visited on 2016-06-22, [Online]. Available: http://akizukidenshi.com/download/ds/senba/GL55%20Series%20Photoresistor.pdf.

[22] Indian Standards Institution. (1992). IS 3646 Part I: Code of Practice for Interior Illumination, [Online]. Available: https://law.resource.org/pub/in/bis/S05/is.3646.1.1992.pdf.

[23] Seeedstudio. (2016). Grove - PIR Motion Sensor. visited on 2016-06-21, [Online]. Available: http://www.seeedstudio.com/wiki/Grove_-_PIR_Motion_Sensor.

[24] Pi4J. (2016). Pi4J library. visited on 2016-06-21, [Online]. Available: http://pi4j.com/.