universität wien

# DIPLOMARBEIT / DIPLOMA THESIS

Titel der Diplomarbeit / Title of the Diploma Thesis

**„An IoT based Monitoring System using Raspberry Pis and Seattle Testbed"**

verfasst von / submitted by

**Peter Klosowski**

angestrebter akademischer Grad / in partial fulfillment of the requirements for the degree of

**Bachelor of Science**

Vienna, 2018

| | |
|---|---|
| Studienkennzahl lt. Studienblatt / degree programme code as it appears on student record sheet: | A 033 521 |
| Dissertationsgebiet lt. Studienblatt / field of study as it appears on student record sheet: | Computer Science |
| Betreut von / Supervisor: | Univ.-Prof. Dipl.-Math. Peter Reichl, M.A. St. |
| Gutachter: | Dr. Bakk. (FH) Albert Rafetseder, BSc MSc |

Ich versichere an Eides statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Peter Klosowski, Vienna, Wednesday 21st February, 2018

# Abstract

Single board computers like the Raspberry Pi have been popular for quite some time now. More and more Internet of Things (IoT) devices are pushed onto the market. This bachelor thesis proposes an approach to built a decentralized environmental monitoring system that is flexible regarding sensor compatibility. Using Seattle Testbed as a code base, allowed to gather sensor data over the Internet with access control. The Restricted Python (Repy) application programming interface (API) was extended with compatible drivers for the sensors. To visualize that data, a homepage was developed using modern plotting tools like D3.

# Contents

# Figures

# Tables

# Acronyms

**API** application programming interface. I, 12, 14
**C** Celsius. 9, 16, 17
**$CO_2$** carbon dioxide. 10
**CoSy** Cooperative Systems. III, 4
**CPU** Central Processing Unit. 2, 7
**$eCO_2$** estimated carbon dioxide. 8, 10, 15, 16, 19
**EEML** Extended Environments Markup Language. 3
**FLOSS** Free/Libre Open Source Software. 1
**GCC** GNU Compiler Collection. 12
**GPIO** general purpose input/output. 1
**GUI** graphical user interface. 3
**hPa** hectopascal. 9, 16, 17
**$I^2C$** Inter-Integrated Circuit. 1, 5, 9, 10, 12
**IoT** Internet of Things. I, III, 1, 3–9, 15, 16, 20
**IR** infrared. 1, 8, 10, 15, 16, 18
**IT** Information Technology. 1
**JSON** JavaScript Object Notation. 13
**mm** millimeter. 1

**nm** nanometer. 1  
**PCB** printed circuit board. 9  
**ppb** parts per billion. 10  
**ppm** parts per million. 10  
**Repy** Restricted Python. I, 10, 12, 15  
**SB** sandbox. 2, 12, 20  
**SCL** serial clock. 1  
**SDA** serial data. 1  
**seash** Seattle Shell. 8, 12, 15, 16, 20  
**TVOC** total volatile organic compounds. 8, 10, 15, 16, 19  
**UML** Unified Modeling Language. 4  
**UV** ultraviolet. 1, 8, 10, 15, 16, 18  
**VM** Virtual Machine. 7, 8

# 1 Motivation and Theoretical Background

Over the years, since the release of the Raspberry Pi [Ras18], a low-cost single-board computer, a whole group of hobbyists, academics, companies, even hackers and IT criminals grabbed onto this trend of IoT and started to experiment. They built useful and unique devices with it. One field that is widely explored is the monitoring of various environments. Using sensors to expand the application of the Raspberry Pi, it is possible to cover a lot of different use cases, e. g. connecting a camera to observe the surroundings.

But what is this IoT which gained momentum in the last few years? IoT is a term for every object that gets hooked up to the Internet through embedded systems like the Raspberry Pi. They should make lives easier and improve the live quality in the best case scenario by offering new functions to older devices and even control them when nobody is near them. On the other hand they need to be developed with security in mind so that attackers do not get access to them. [Xia+12]

The sensors used for this project were collecting various kinds of data. The light sensor was catching visible and infrared (IR) light. Visible light has a wavelength of 400 to 770 nanometer (nm) and as the name states it is visible to humans. Shorter than that wavelength is the ultraviolet (UV) light with 1 to 400 nm. It is that light that hurts the human skin and causes skin cancer in the worst case scenario. Light with a wavelength longer than 760 nm is called IR light. It can go up to 1 millimeter (mm) and the human perceives it as heat. [MNH03]

Those sensors can be connected through the general purpose input/output (GPIO) pins. Some of them are capable of utilizing the Inter-Integrated Circuit ($I^2C$) protocol that allows multiple connections of peripherals. Newer versions of the Raspberry Pi (starting with Revision B) are using bus one and the GPIO pins three (serial data (SDA)) and five (serial clock (SCL)) to exchange messages with the sensors. Each connected slave has its own address that is up to 7 bit long. It is worth mentioning that two slaves cannot have the same address because it would conflict with messaging the right one. [Gay14, p. 167ff]

In this work a Raspberry Pi 2B was combined with three different sensors. Knowing that the finished box would be used by other academics and students for their projects it was necessary to meet some requirements. Ideally the box should not be too expensive so multiple ones can be produced and set up. To do efficient experiments a decentralized approach was essential in order to be able to deploy them for different scenarios and use cases. The execution of specific software code on multiple boxes to collect data was therefore necessary. Depending on what kind of data collection was done, different code snippets would be used and uploaded to the boxes. They should be in sync even if the units do not know of each other. Another point was to build a Free/Libre Open Source

Software (FLOSS) so it is accessible by other interested people in the future and can be used to enrich their projects. Open data was an important aspect as well. If desired it should be possible to upload all collected data and present it to people who observe the development of it and the project for example.

Regarding cluster management, software updates or code pushing a software solution was used called Seattle. Seattle describes itself as ”a free, open-source platform for networking and distributed systems research”. Seattle runs on many devices and allows researchers and students for example to access a whole range of those machines that are connected to the network through the Internet. They are limited though, and are just allowed to work inside a sandbox (SB) that limits the use of Central Processing Unit (CPU), memory and storage space. [Sea18b]

## 1.1 *Application of the Box*

As mentioned before, the field of use would be a more academic one but that does not really say a lot about its practical use in the end. It was already clear in the beginning, that a more general approach should be taken. Therefore, a quick replacement of one or more sensors would be necessary at some point.

One idea for a student project would be to use moisture, temperature, light and water quality sensors, pair them with the Raspberry Pis and deploy all of it with the software build to measure the soil in a garden or perhaps even a park or golf course. But even in an apartment it would be possible to use that kind of technology to improve the health of home plants.

Another way of use would be to monitor the well being of persons in an apartment by making sure that air quality, temperature, humidity and pressure sensors are in the green. If one of those values would indicate a bad influence to the health or comfort of the home owner, then it would be a good idea to take countermeasures.

# 2 Related Work

Roselle B. Anire et al [ACA17] focused on monitoring soil in a greenhouse with four Raspberry Pis that were connected together over the network and sending data to a PC that aggregated it. Every 60 minutes new data come in and are saved in an excel file. A simple graphical user interface (GUI) was created to visualize this data.

Reinfurt L. et al [Rei+16] gathered five different state of the art design patterns, that have been identified by reviewing existing products. They identify the problems that occur during implementations and also offer a satisfying solution.

Shete R. and Agrawal S. [SA16] presented a low cost urban climate monitoring system using a Raspberry Pi and various sensors like air quality, light and temperature. The collected data is uploaded to Adafruit IO, a cloud IoT system, via WiFi. Through an MQTT Broker the data is distributed to subscribed user.

Ibrahim M. et al [Ibr+15] collected environmental data using a Raspberry Pi and a few sensors connected to it. Even a sensor registering earthquakes was installed. The protocol Extended Environments Markup Language (EEML) was used to share the sensor data with another remote device like a laptop, smartphone or a web service in the cloud.

Wixted A. et al [Wix+16] presented LoRa and the LoRaWAN protocol and compared it to other state of the art technologies that are used for IoT. The protocol is explained and evaluated on reliability and performance with a positive outcome.

# 3 System Design and Solution Concept

Based on the requirements discussed in the introduction of this work, a software design was developed. The whole documentation that is partially described here can also be retrieved from the personal GitHub repository[1]. Regarding the following content, it is worth mentioning that the already existing Seattle Testbed [Sea18a] was used as a codebase. That affected some of the design decisions as a result.

All of the following system designs were modeled with the Unified Modeling Language (UML) standard. It is widely used in software engineering and allows a quick understanding of high and low level structures with the provided tools. It is also capable of creating business and similar models, but these were not necessary this time. [Obj17]

## 3.1 *System Overview - Use Case*

At the beginning of this project the future features were defined using a Use Case diagram. Visible in Figure 1 are six use cases that made the final cut. There are two actors that will be using them. In this case the "User" will be anybody that has access to the system and is allowed to operate with it. The "IoT-Box" is the combination of a Raspberry Pi, sensors and the Seattle software build.

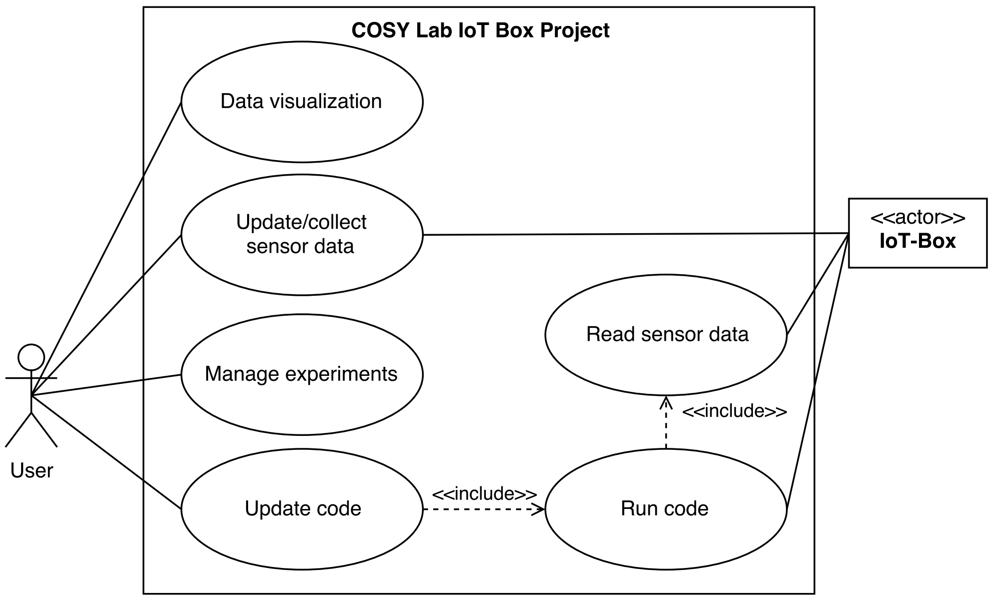


Figure 1: Use case diagram describing the CoSy Lab IoT-Box Project

1 https://github.com/Peter0014/COSY-Lab-IoT-Box/tree/master/DesignDoc

### 3.1.1 *Manage Experiments*

This use case includes two work flows regarding experiments that are run on one or several IoT-Boxes. Experiments in this scope are all installations of the software on the Raspberry Pis. It should be possible to register new experiments and remove ones that are not needed anymore. Also a secure access should be guaranteed in case somebody that is not authorized tries to access this information. In this case the job is completely done by Seattle.

### 3.1.2 *Update Code*

To adapt to different use cases of the boxes, it will be necessary to update its purpose and code over the Internet. The user is able to upload a code snippet with the functions that he wants to run and decides to push it to one or multiple boxes. Here the user has the choice which sensor data are collected, the interval and the file format it is saved in. After a successful upload the boxes will try to run it and start its data collection or return an error to the user. Access control should also be implemented in this case to allow just authorized users to update code fragments. Seattle also has methods that help with that whole use case.

### 3.1.3 *Run Code*

Every uploaded code needs to be run by the boxes. It is important to do that so the user is able to collect data from the sensors attached to the box. Depending on the code that is pushed different kind of sensor data can be fetched and saved. This job gets done by Seattle as well.

### 3.1.4 *Read Sensor Data*

To analyze data they need to be collected first. Over the $I^2C$ protocol a connection will be established and some sensor data will be fetched. The uploaded code decides which sensors are contacted and the time interval in which new data is appended to a file. Depending on the use case it should be possible to use different sensors that were not thought of at the beginning of this work or are good additions to the existing ones. Because every sensor is different, like newer revisions or different kinds of design and companies, it will be mostly necessary to update the software loaded onto the Raspberry Pis. This process is explained in more detail in Section 4.

### 3.1.5 *Collect Sensor Data*

It is possible to download the files over the Internet produced by the user supplied code and after some sensor data were read and saved to the boxes. The data that were fetched can be used to plot some graphs using the dedicated homepage that was developed during this work and is described in Section 4 and the next use case. Regarding the open data requirement it should be able to publish that data for everybody to download, use, share and work with.

### 3.1.6 *Data Visualization*

The data visualization is another crucial part of this work. It would also be possible to visualize the data with applications like Excel from Microsoft[2] and similar tools. It would be just necessary to provide the raw data for the end user. On the other hand it is handy to offer the user a tool, or web page in this case, where he can upload his data and the visualization is automatized. Additionally it would be possible to store data on the web server and demonstrate it using the plotting tool that is specially customized to the format and output of the boxes.

### 3.1.7 *Alternatives*

Searching for solutions, it was decided that Seattle was a satisfying approach in this case. There were other thoughts and alternatives before Seattle came up though. For example it was an option to built the decentralized system behind it from scratch. Using IoT patterns mentioned in [Rei+16], like device shadowing for a persistent storage of the data, that would have allowed to reach a device, even if it is offline and unavailable right now. Implementing a rules engine would have helped to dynamically run routines for fetching sensor data if the user sends a particular message to the system. If the boxes would have been used in outdoor places that are not overlooked or secured, it would also have been helpful to have a remote lock feature.

Some of these patterns are already available in the Seattle framework, like a rules engine that supports actions via a special scripting language. Additionally to that, it also covers and solves problems like cluster management and code pushing. It was also decided that building an IoT system from scratch would be out of scope for a bachelor project.

## 3.2 *System Overview - Software Deployment*

The deployment of the software artifacts is divided into three parts. Looking at Figure 2 it can be differentiated between four different devices:

"User-Client" is typically the home computer of the end user

"Web-Server" is used for the visualization of the data

2 https://products.office.com/de-at/excel

"Raspberry Pi" is part of the subsystem "IoT-Box" and executes the user provided code

"Various Sensors" are part of the subsystem "IoT-Box" and collect data from the environment

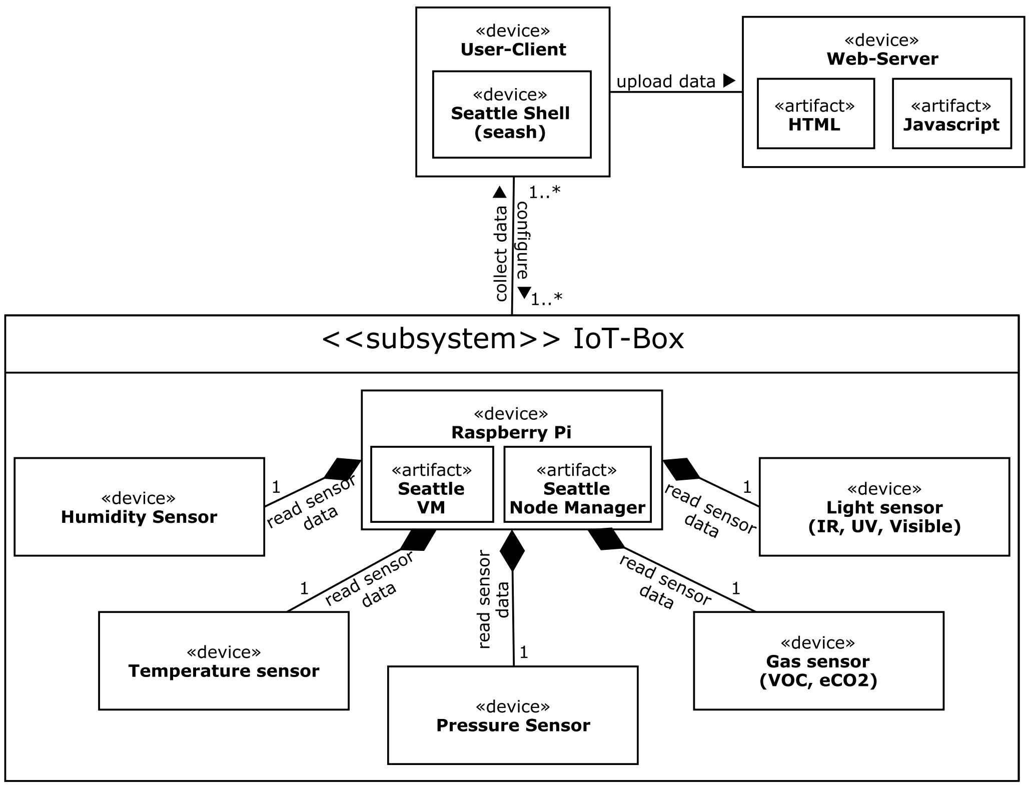


Figure 2: Deployment diagram

### 3.2.1 *Raspberrry Pi*

The Raspberry Pi includes parts of the Seattle framework. The Seattle Virtual Machine (VM) executes the user provided code and prevents to run malicious content that could break out of the VM to get access to the system it is running on. It also prevents the application from using too many resources from components like the CPU or memory. [ZRC13, p. 38]

The Seattle Node Manager is important for quite a few things. First, it manages access control and allows just authorized parties to execute code on the device it is installed on. At the same time, it provides an interface to upload code, manage a VM and download collected data and logs from it. [ZRC13, p. 38]

All kinds of different sensors joined with a Raspberry Pi are components of the subsystem "IoT-Box". In this case the sensors are a humidity, temperature and pressure sensor, a gas sensor collecting total volatile organic compounds (TVOC) and estimated carbon dioxide ($eCO_2$) data, as well as a light sensor for IR, UV and visible light that are described further in Section 4.1. Using Seattle allows to have one or many IoT-Boxes like this that are configured by one or many authorized User-Clients.

### 3.2.2 *Web-Server*

The visualization of the data takes place on the Web-Server, or to be precise in fact on the User-Client by processing the JavaScript files that are provided by the server. It is possible to load a file with the sensor data into the script and let it plot different graphs for every available data entry. The output is presented to the user through the web page in his browser. Regarding open data it is also possible to save files on the web server by the administrator to share it with the public.

### 3.2.3 *User-Client*

Through Seattle Shell (seash) it is possible to connect to Raspberry Pis if the user is authorized to do so. The User is able to install seash onto his device and configure the VM, push code to it, as well as collect the data that are generated through his code. Seash operates here as a service manager and allows the interaction with the node manager. The user client also connects to the web server via a browser of his choice and opens up the web page to process his data, that he downloaded from the Raspberry Pis. [ZRC13, p. 38]

# 4 Implementation

There are three parts to the implementation. The first part was the decision of buying several sensors, solder them if necessary, connect them to the Raspberry Pi and test them. The second part consisted of developing the software to operate the sensors and extend Seattle in that way, that it can call those software functions. In the end a home page was built that can be used for visualization and informing other people about the project and progress.

## 4.1 *Sensor Exploring*

Looking for different use cases that were not just practical but also useful in the sense of doing research with them, the decision was made to buy three different sensors. In combination they were collecting useful data about the environment that could be used in all rooms of a home, in a lab and outdoors for example. Furthermore all sensors are breakout boards by the company Adafruit[3] and it was necessary to solder the header onto the printed circuit board (PCB). It was also required that all sensors support the $I^2C$ protocol.

### 4.1.1 *Temperature, Barometric Pressure and Humidity Sensor*

The first thing that comes to mind when talking about environment is the temperature. After all it is something that we experience ourselves everyday. Combined with measuring barometic pressure, also known as atmospheric pressure, and humidity it is a suitable addition to the IoT-Box. The BME280 environmental sensor was manufactured by Bosch[4] and has an accuracy of ±3% regarding humidity, ±1 hectopascal (hPa) with barometic pressure and ±1.0°Celsius (C) with temperature. [Ada18a]

Additionally, The company STMicroelectronics[5] sponsored sensors from their product line. For one, the HTS221 for relative humidity and temperature and the LPS25HB for absolute pressure was sent to support the project. They have an accuracy of ±0.5°C regarding temperature between 15°C and 40°C, ±3.5% with humidity and ±1 hPa with barometic pressure. [STM18a] [STM18b]

3 https://www.adafruit.com/
4 https://www.bosch-sensortec.com/
5 http://www.st.com/

### 4.1.2 *Light Sensor*

Knowing the UV index is sometimes quite a useful information, like when you're outside a lot on a hot summer day or when you just want to do some measurements. The SI1145 light sensor from SiLabs[6] calculates the UV index based on the IR and visible light from the sun. As a side note, these last two values have no units to them, because they are based on the light that is falling onto them. [Ada18c]

### 4.1.3 *Air Quality Sensor*

There is a lot of air pollution in big cities that is harmful to every person if it is present in high amounts. Especially in the morning when people drive to work with cars the carbon dioxide ($CO_2$) pollution is increasing immensely [CD11]. To measure that, it was decided to get an air quality sensor - the CCS811. The chip was developed by the company ams[7] and collects $eCO_2$ and TVOC data. It has a range from 400 to 8192 parts per million (ppm) regarding $eCO_2$ and 0 to 1187 parts per billion (ppb) with TVOC. It is also recommended to run it 48 hours after buying it to burn it in and always start to read data 20 minutes after beginning the sensor reading. Unfortunately, after starting the data collection for the evaluation it was clear that the air quality sensor does not work properly with the Raspberry Pi because the sensor uses clock stretching that the Pi does not support. [Ada18b]

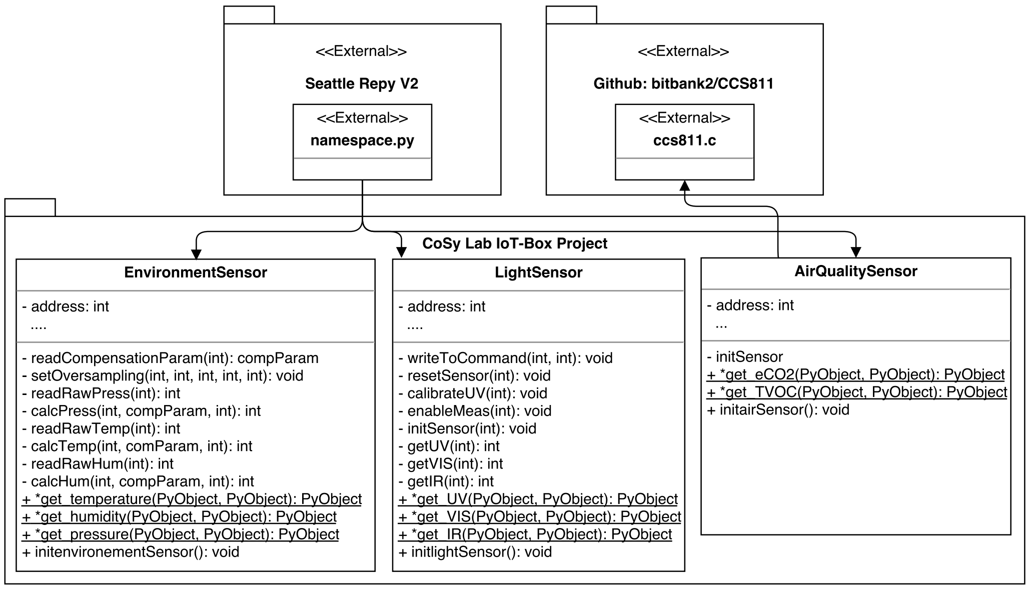
## 4.2 *Software Development*

The next step taken after connecting the sensors to the Raspberry Pi over the $I^2C$ dedicated Pins was to develop software classes and extend the Seattle Repy V2 code to be able to call the functions in these classes. Figure 3 shows a small excerpt of these classes that are described in the following sections as well.

### 4.2.1 *Writing the Classes*

At the first meeting there was a discussion about the programming language that would be used for this project. The decision was between Python[8], Java[9] and C. Seattle is programmed using the Python language. It was therefore a good choice for this situation. Consequently, it was inevitable to use a wrapper if Java or C code would be used to extend the existing Seattle code base.

6 https://www.silabs.com/
7 http://ams.com/
8 https://www.python.org/
9 https://java.com/



<<External>>
Seattle Repy V2
<<External>>
namespace.py
<<External>>
Github: bitbank2/CCS811
<<External>>
ccs811.c
CoSy Lab IoT-Box Project
EnvironmentSensor
- address: int
....
- readCompensationParam(int): compParam
- setOversampling(int, int, int, int, int): void
- readRawPress(int): int
- calcPress(int, compParam, int): int
- readRawTemp(int): int
- calcTemp(int, comParam, int): int
- readRawHum(int): int
- calcHum(int, compParam, int): int
+ *get_temperature(PyObject, PyObject): PyObject
+ *get_humidity(PyObject, PyObject): PyObject
+ *get_pressure(PyObject, PyObject): PyObject
+ initenvironementSensor(): void
LightSensor
- address: int
....
- writeToCommand(int, int): void
- resetSensor(int): void
- calibrateUV(int): void
- enableMeas(int): void
- initSensor(int): void
- getUV(int): int
- getVIS(int): int
- getIR(int): int
+ *get_UV(PyObject, PyObject): PyObject
+ *get_VIS(PyObject, PyObject): PyObject
+ *get_IR(PyObject, PyObject): PyObject
+ initlightSensor(): void
AirQualitySensor
- address: int
...
- initSensor
+ *get_eCO2(PyObject, PyObject): PyObject
+ *get_TVOC(PyObject, PyObject): PyObject
+ initairSensor(): void

Figure 3: Excerpt of the class diagram

The "Computer Language Benchmark Game" made it their goal to benchmark different programming languages with several programs. Looking at the performance at [The18], then it becomes clear that C is the fastest of all these languages. Not far after that, there is Java with less than three seconds difference. Far behind is Python with ten and up to almost 100 seconds run time. After all, it depends on the program that is being executed. Some programming languages do the job more quickly and effective than others. In [An06, p. 54], there is a runtime comparison between Python, C and Python extended with C. There is a big improvement of the factor 25 (431.42/17.05) between just Python and the one extended by C if the program from [An06] is run 500 times.

The decision was also made based on a personal preference. Being more in contact with Java and C through internships and courses at the university, it was easier to perform this project with the already available know-how. Looking for available resources online, there was an official section in the documentation of Python about extending it with C[10], as well.

### 4.2.2 *Extending Seattle with C*

On the basis of the previous information it was an appropriate solution to go with the programming language C. Drivers were written with the help of the documentations from the sensors that were attached to the Raspberry Pi and resources found on the Internet. After testing them on the device itself and checking that the right values were

10 https://docs.python.org/2/extending/extending.html

returned, a shared library and Python module was created using the GNU Compiler Collection (GCC). For example, a shell command could look like that:

```
gcc -shared -Xlinker -export-dynamic -o environmentSensor.so
-I/usr/include/python2.7/ -lpython2.7 -lwiringPi
BME280_TempSensor.c
```

Listing 1: build command for a shared library

In this case a shared library is created with the parameters "-shared -Xlinker -export-dynamic". The output file would be "environmentSensor.so" and the input file is the C class "BME280_TempSensor.c". The WiringPi and Python libraries and imports are necessary to build the class properly. The .so file, that is produced calling that command, can be loaded into Python as a module now. Visible are just special functions though. Looking at Figure 3, the static functions that return the PyObject are added in the init function (like initenvironmentSensor()) so that Python is able to work with them.

Having a working Python module available, it was now possible to extend Seattle with that code. The modules were added to the Repy SB by expanding the API with the specific function calls, like get_temperature. This was done in the namespace.py file (see Figure 3). The extended Repy V2 source code is available online at [Rep18].

All of that work allows the user in the end to call the functions through seash over the Internet. The message flow when running the Repy code provided by the user is illustrated in Figure 4. Here the experiment calls the API in the Repy SB that runs the Python code. At that point, it was possible to call a Python module, a less preferred way to execute Popen to use Shell-Tools or in this case through the CPython API the C code is executed and accesses the I$^2$C messages through the wiringPi library. The values are then returned to the experiment where it is being processed.

## 4.3 *Visualization and Homepage*

The last part of the implementation was the homepage to visualize the data. The goal of the homepage was to provide the user with a way to visualize his data and to have a platform to inform about other information regarding the boxes as well as developments.

The website, that is visible in Figure 5, uses various libraries and frameworks to offer those features. First and foremost the design and front-end was accomplished using Bootstrap[11], dependent on that JQuery[12] is necessary, too, and last but not least Font Awesome[13] for various kinds of icons. Since Bootstrap is able to optimize a web page for both mobile and desktops, it is able to view this homepage on most modern devices

11 https://getbootstrap.com/
12 https://jquery.com/
13 https://fontawesome.com/

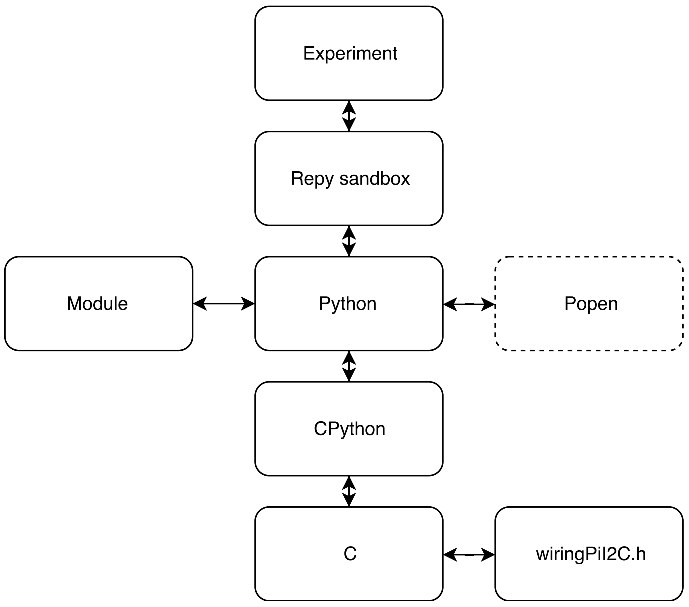


Figure 4: Path from experiment to C code

with multiple screen resolutions. As a side note the homepage is just there to showcase the visualization and provide a template for others therefore there is just one web page available as of now.

To plot the charts the JavaScript library Angular-nvD3[14] was used. This library depends on a few other ones. As the name states Angular[15], NVD3[16] and D3.js[17] are required to be loaded, too. They are the back-end of this homepage and are responsible to build the charts using the data uploaded by the user.

First the user is asked to upload a file. That file needs to be in JavaScript Object Notation (JSON) format[18]. The function that processes the file searches for a specific key and adds the value and date to the corresponding chart. The available keys that were used in this project are:

- Time
- Temperature
- Humidity
- Pressure

14 http://krispo.github.io/angular-nvd3/
15 https://angular.io/
16 http://nvd3.org/
17 https://d3js.org/
18 https://www.json.org/

- UV
- IR
- VIS
- eCO2
- TVOC

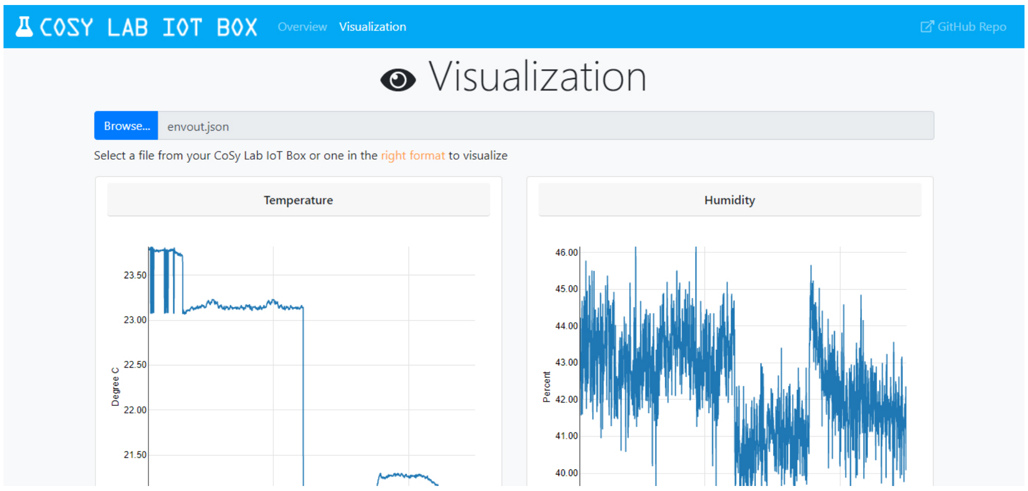


Figure 5: Screenshot of the visualization homepage

The whole collection of key-value pairs is iterated and pushed into the values array of the charts. The charts are refreshed through the API from Angular-nvD3 after the last task is finished. That process after uploading the file normally takes just a few seconds and depends on the amount of values being inserted.

# 5 Evaluation and Testing

Many use cases are possible with the IoT-Box and there are plenty of possibilities to extend it with available sensors on the market. Therefore it is not easy to test the box for every eventuality but at least some evaluation and testing can be done for the most probable utilization of it.

## 5.1 *Functional Evaluation*

Working with the box for the last weeks and months there were some work flows that repeated themselves quite a lot. Based on that a use case scenario was created:

#### ***Use Case Scenario - Collecting Data and Visualizing it***

- John Doe successfully connects over seash with his cluster of IoT-Boxes with his personal key
- John prepared a Repy code that he now uploads to the boxes and starts them
- After a while John stops the execution of his code and downloads the files that were created during that time
- Fortunately, John saved all those collected values in one JSON file and has now temperature, humidity, pressure, UV, IR, visible light, $eCO_2$ and TVOC data that he uploads to the web page
- The web page processes his file and plots graphs for them that John can analyze

Some tests were conducted on the basis of this use case scenario that are visible in Table 1. Not everything was tested because some functions are already implemented in Seattle and were not worked on as part of this work, like up- and downloading files to the boxes for example.

Most of the tests worked well and satisfying. The partial result with the long-term experiment is connected with the other two partial results. As mentioned in Section 4.1.3 the air quality sensor does not fully work with the Raspberry Pi. It was just possible to collect data for a shorter period of time than with the others. The other sensors worked properly as we see in Table 1. A long-term experiment is therefore possible if the sensor is not used.

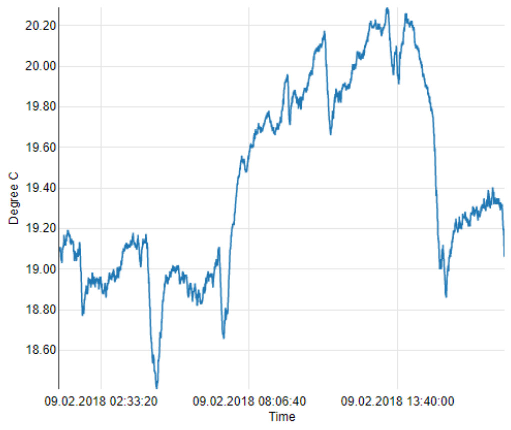
| **Test** | **Result** |
|---|---|
| Doing a long-term experiment | Partial |
| Reading temperature | Worked |
| Reading humidity values | Worked |
| Reading pressure values | Worked |
| Reading UV values | Worked |
| Reading IR values | Worked |
| Reading visible light values | Worked |
| Reading $eCO_2$ values | Partial |
| Reading TVOC values | Partial |
| Visualizing collected data | Worked |

Table 1: Tested features of the IoT-Box implementation

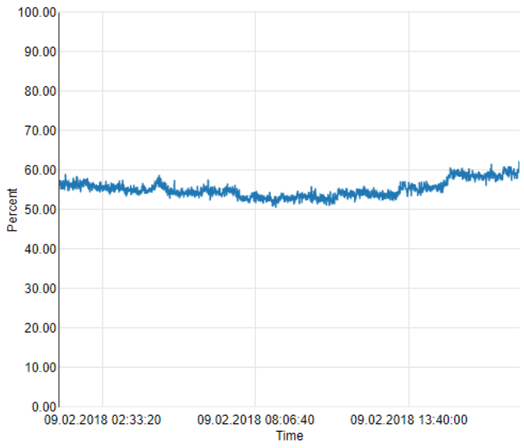
## 5.2 *Experiment Evaluation*

For the experiment the IoT-Box was put at the window during winter time in Vienna, on the 09.02.2018 to be exact. According to WetterOnline[19], the outdoor temperature at ”Hohe Warte” was between 0°C and 3°C. humidity was measured at 74% and pressure at 1019 hPa. The IoT-Box collected data for several hours before the files were downloaded through seash. Having the file at hand, it was loaded into the visualization web page to create some charts.

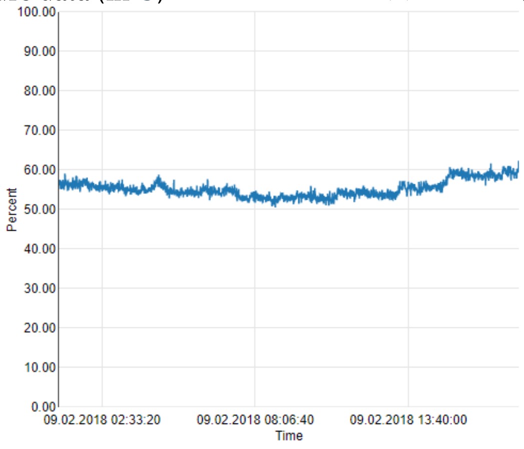
19 https://www.wetteronline.at/wetterdaten/wien



(a) Temperature data (in C)



(b) Humidity data (in %)



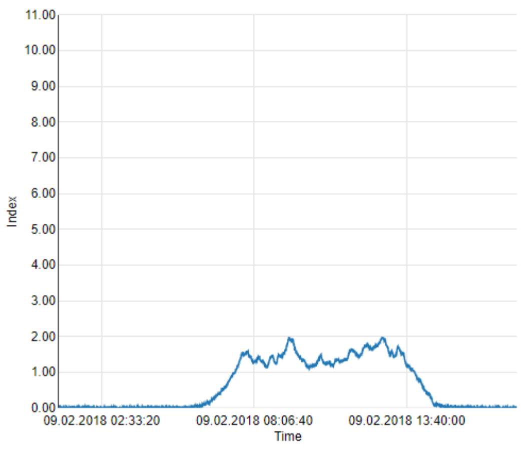
(c) Pressure data (in hPa)

Figure 6: Data collected by the BME280 sensor

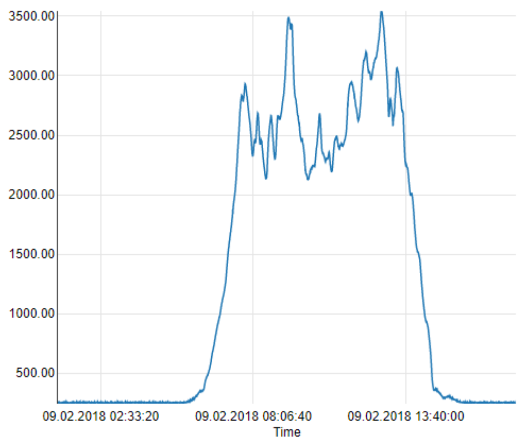
First we have the temperature in Figure 6a. At 7 in the morning the heating started and the temperature has risen up to its peak at 13:15 with 20.30°C. At 15 the window was opened for 30 minutes and the temperature fell 2°C before it rises up again a bit. The lowest point was reached at 4:40 am with 18.42°C. The mean is 19.43 from all 2000 collected values.

The humidity in 6b did not change much. It started at midnight with 56.89% humidity and in the end just raised by less than 6% at 62.34%, its peak. The lowest point was at 7:50 am with 50.55% and the mean is 55.24%. Opening the window also pumped up the humidity a small amount to compensate the change between outside and the room.

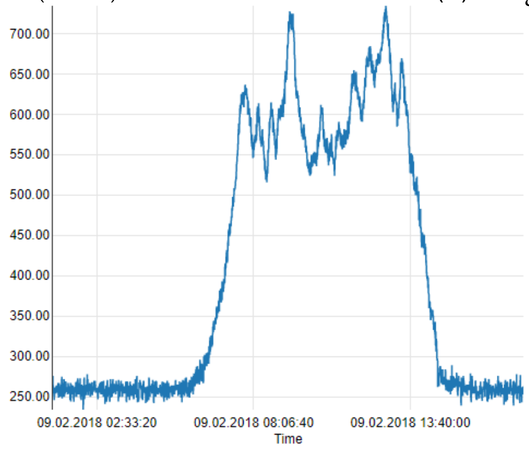
Comparing the pressure with the information from WetterOnline it is apparent that it was climbing that day. That also reflects with the values recorded by the box. Starting at 979.39 hPa that numbers grows up to 985.77 hPa.



(a) UV data (index)



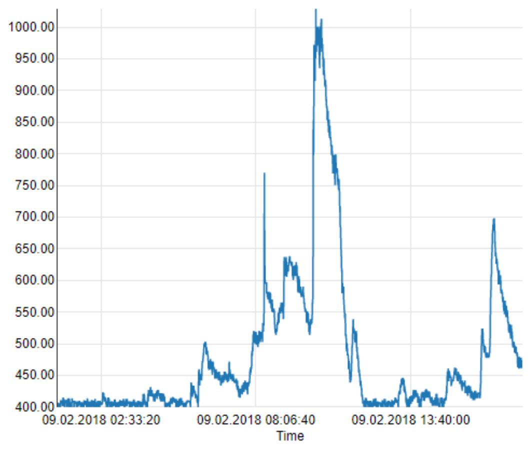
(b) IR light data



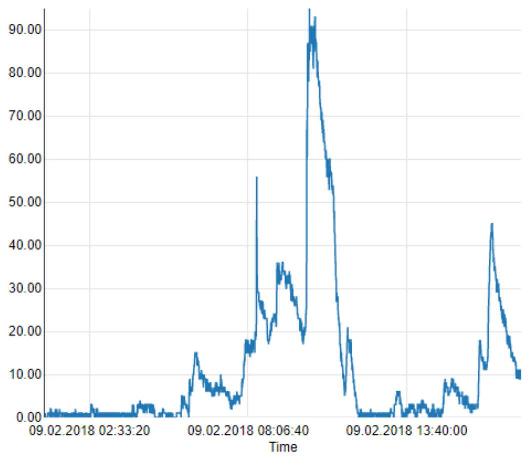
(c) Visible light data

Figure 7: Data collected by the SI1145 sensor

That day the sun did not shine very bright. Inspecting Figure 7b and 7c we can see that the sunrise was at 7 in the morning. Looking at the graphs we can see that there are no units because the sensor just measures how many light falls into it. Therefore it is just there for information but using those two values it is possible to calculate the UV index that we see in Figure 7a. That day it stayed under two. That is a pretty low value and not harmful for the normal human skin.



(a) $eCO_2$ data



(b) TVOC data

Figure 8: Data collected by the CCS811 sensor

# 6 Conclusion

The solution approach presented in this work is combining Seattle with the Raspberry Pi to build a network of IoT devices and grab the sensor data off them that they were collecting. After working with that setup for several weeks, it emerged, that the advantages of Seattle with its cluster management, code pushing, the SB and extendability of the API with just a few lines of code made it the suitable candidate as a software base. The flexibility of that approach made that solution more future proof because new sensors can be attached and used with the existing ones.

During the implementation there were no bigger problems that came up. The only negative development was the sensor not working properly with the Raspberry Pi. As already described in the functional evaluation, it was still possible to operate with it but not in the way it was intended. That was a small drawback regarding the full functionality of the IoT-Box but it is fixable quite quickly by replacing the sensor with another one.

Additionally, it is just possible to configure the IoT-Boxes over the Internet. They need a working connection so that the user can upload and start his file through seash. On the other hand it also simplifies the process of configuring multiple boxes at once.

## 6.1 *Future Work*

In the future it would be possible to expand the sensor diversity. There are a lot of different sensors for sale like motion, sound, GPS and plenty more. Even sensors that have not been developed yet and are just future thinking could possibly be paired with a Raspberry Pi which has Seattle installed.

To conduct some outdoor research, it would be helpful to have a support for LoRa and LoRaWAN [Sor+15]. LoRaWAN is based on LoRa and is able to store data in a database over a long distance by uploading them through a gateway. An application server like the visualization web page for example could fetch the data and have an up-to-date collection of charts.

# References

[ACA17] R. B. Anire, F. R. G. Cruz, and I. C. Agulto. "Environmental wireless sensor network using raspberry Pi 3 for greenhouse monitoring system". In: *2017IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM).* Dec. 2017.

[Ada18a] Adafruit - BME280. https://learn.adafruit.com/adafruit-bme280-humidity-barometric-pressure-temperature-sensor-breakout?view=all. accessed February 10, 2018.

[Ada18b] Adafruit - CCS811. https://learn.adafruit.com/adafruit-ccs811-air-quality-sensor?view=all. accessed February 10, 2018.

[Ada18c] Adafruit - SI1145. https://learn.adafruit.com/adafruit-si1145-breakout-board-uv-ir-visible-sensor?view=all. accessed February 10, 2018.

[An06] F. An. "Efficient parallel programming using Python and C". PhD thesis. Oklahoma State University, 2006.

[CD11] K. Caldeira and S. J. Davis. "Accounting for carbon dioxide emissions: A matter of time". In: *Proceedings of the National Academy of Sciences* 108.21 (2011), pp. 8533–8534.

[Gay14] W. Gay. *Mastering the Raspberry Pi.* Apress, 2014. ISBN: 978-1-4842-0181-7.

[Ibr+15] M. Ibrahim, A. Elgamri, S. Babiker, and A. Mohamed. "Internet of things based smart environmental monitoring using the Raspberry-Pi computer". In: *2015 Fifth International Conference on Digital Information Processing and Communications (ICDIPC).* Oct. 2015.

[MNH03] K. Mori, T. Nishida, and H. Hashimoto. *Ultraviolet light measuring chip and ultraviolet light sensor using the same.* US Patent 6,551,493. Apr. 2003.

[Obj17] Object Management Group. *Unified Modeling Language.* https://www.omg.org/spec/UML/2.5.1/PDF. Dec. 2017.

[Ras18] Raspberry Pi Foundation. https://www.raspberrypi.org. 2012 (accessed February 5, 2018).

[Rei+16] L. Reinfurt, U. Breitenbücher, M. Falkenthal, F. Leymann, and A. Riegg. "Internet of things patterns". In: *Proceedings of the 21st European Conference on Pattern Languages of Programs.* ACM. 2016, p. 5.

[Rep18] Repy V2 - Github Fork. https://github.com/Peter0014/repy_v2. 2017 (accessed February 9, 2018).

[SA16] R. Shete and S. Agrawal. "IoT based urban climate monitoring using Raspberry Pi". In: *2016 International Conference on Communication and Signal Processing (ICCSP).* Apr. 2016.

[Sea18a] Seattle. https://seattle.poly.edu/html/. 2013 (accessed February 5, 2018).

[Sea18b] Seattle - Github. https://github.com/SeattleTestbed. 2014 (accessed February 5, 2018).

[Sor+15] N. Sornin, M. Luis, T. Eirich, T. Kramp, and O. Hersent. *LoRaWAN™ Specification.* LoRa Alliance. 2015.

[STM18a] STMicroelectronics - HTS221. http://www.st.com/content/st_com/en/products/mems-and-sensors/humidity-sensors/hts221.html. accessed February 13, 2018.

[STM18b] STMicroelectronics - LPS25HB. http://www.st.com/content/st_com/en/products/mems-and-sensors/pressure-sensors/lps25hb.html. accessed February 13, 2018.

[The18] The Computer Language Benchmarks Game. http://benchmarksgame.alioth.debian.org/u64q/which-programs-are-fastest.html. accessed February 15, 2018.

[Wix+16] A. J. Wixted, P. Kinnaird, H. Larijani, A. Tait, A. Ahmadinia, and N. Strachan. "Evaluation of LoRa and LoRaWAN for wireless sensor networks". In: *2016 IEEE SENSORS.* Oct. 2016.

[Xia+12] F. Xia, L. T. Yang, L. Wang, and A. Vinel. “Internet of things”. In: *International Journal of Communication Systems* 25.9 (2012), p. 1101.

[ZRC13] Y. Zhuang, A. Rafetseder, and J. Cappos. “Experience with seattle: A community platform for research and education”. In: *Research and Educational Experiment Workshop (GREE), 2013 Second GENI.* IEEE. 2013, pp. 37–44.