Université Libre de Bruxelles

TRAN-H201-D - Projet multidisciplinaire II - 201617

Smart Thermostat

Intermediary Report

Tutor:

Vansummeren Stijn

Students - Group 3:

Maoujoud David

Marques Correia João

Pereira Acuna Pierre

Roty Adelin

Serroukh Mardi Ilias

Van Heirstraeten Arthur

willaert Pauline

December 21, 2016

Abstract

"Building of a Multi-zone and Smart Thermostat" by Maoujoud David, Marques Correia João, Pereira Acuna Pierre, Roty Adelin, Serroukh Mardi Ilias, Van Heirstraeten Arthur, Willaert Pauline, Université Libre de Bruxelles, Academic year 2016-2017.

This project consists in the designing and implementing of a multi-zone and smart thermostat, by combining a Raspberry Pi and a NodeMcu communicating via a private Wi-Fi network. An MQTT broker is set up on the Raspberry Pi, allowing for communication in both directions. The thin thermostats are made up of a microcontroller, namely the NodeMcu, supplied with a digital temperature sensor and an infrared sensor that can capture the room temperature and detect human presence. They wirelessly communicate their measurements to a central thermostat unit that consists of a Raspberry Pi, which then stores them in an SQLite database. The central unit also contains a control algorithm that manages thermostatic valve opening for each room.

Key words: multi-zone and smart thermostat, Raspberry Pi, NodeMcu, wireless communication, MQTT broker, temperature and presence sensors, SQLite database, thermostatic valves

Contents

List of Figures

List of Tables

1. http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

1 Introduction

Following the development of small, powerful and affordable technologies, a lot of so-called smart gadgets have surfaced in the past few years.

This project consists in developing one of them, namely a smart thermostat. Similarly to a standard thermostat, its function is to control the temperature of multiple rooms simultaneously, by means of thin thermostats. Furthermore, a smart thermostat has to take into account various parameters, such as outside temperature, human presence, life habits and the heating inertia of the building, in order to effectively heat said building.

A Raspberry Pi and an Arduino-like board are therefore combined and programmed. The board, equipped with a temperature sensor and PIR sensor, is able to detect human presence, measure the room temperature and control the room radiators. The whole structure will make up for a so-called thin thermostat.

The Pi also hosts a database designed to contain past sensor readings, and small local server with an interface. The owners can access all the data and communicate information or commands such as home arrival time to the Raspberry, so the thermostat can preheat the house. This web server is then accessible via a PC web browser.

The following milestones are to be completed throughout the course of the project:

- Milestone 1 : The thin thermostat is operational
- Milestone 2 : The central thermostat is operational
- Milestone 3 : Autonomous communication between the thin thermostat and the central thermostat, as well as a working web interface
- Milestone 4 : The central thermostat is equipped with a smart control algorithm

As of right now, the first two milestones, as well as the the first half of the third milestone, have been completed.

1.1 Hardware

As already stated, the Smart thermostat consists of a Raspberry Pi and a NodeMcu communicating via a private Wi-Fi network. In order to do so, an MQTT broker is set up on the Raspberry Pi. This allows communication in both directions, the NodeMcu sends the readings provided by the temperature and presence sensors to the Raspberry Pi, which then processes them and sends back a response to the NodeMcu for the heating of the room.

Hence, the NodeMcu never does any calculations. The Raspberry, besides processing the data also continuously stores it in a database, and can retrieve previous readings in order to plan the heating schedule.

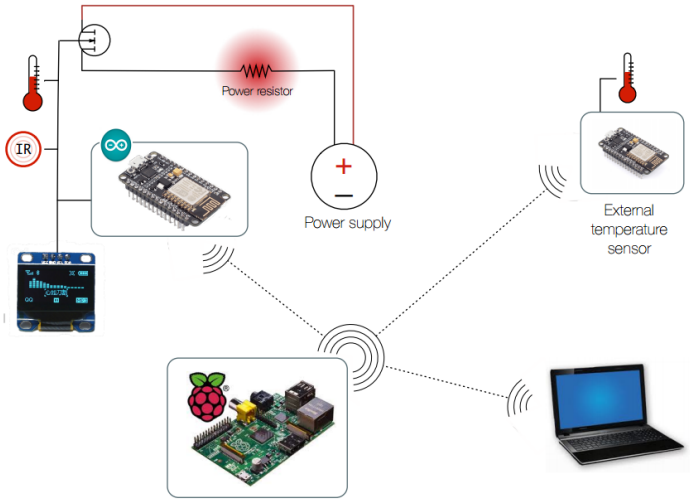


Figure 1 – Simplified Setup

[2]

1.2 Code

It was decided early on to use object-oriented programming to code on the Raspberry Pi, as it provides a clear and modular structure for programs. On the other hand, the Arduino programming language of the NodeMcu, gives no choice as to the type of programming.

Each time a new thin thermostat connects to the wireless network set up by the Raspberry Pi, the Raspberry creates a new instance of this thermostat. This instance, as well as all the ones created before, will then run as threads, allowing all the rooms to be controlled at the same time. The complete UML diagram can be found in Figure 2.

This report is further organized as follows : Section 2 will consist of an explanation of the central thermostat and its underlying features. In Section 3, we will talk about the thin thermostats, in particular their wiring schema and their working principles. The communication between central and thin thermostats will be addressed in Section 4. Following this, Section 5 will present the current state of the project, and the results that we get.

2. Figure from the requirements document

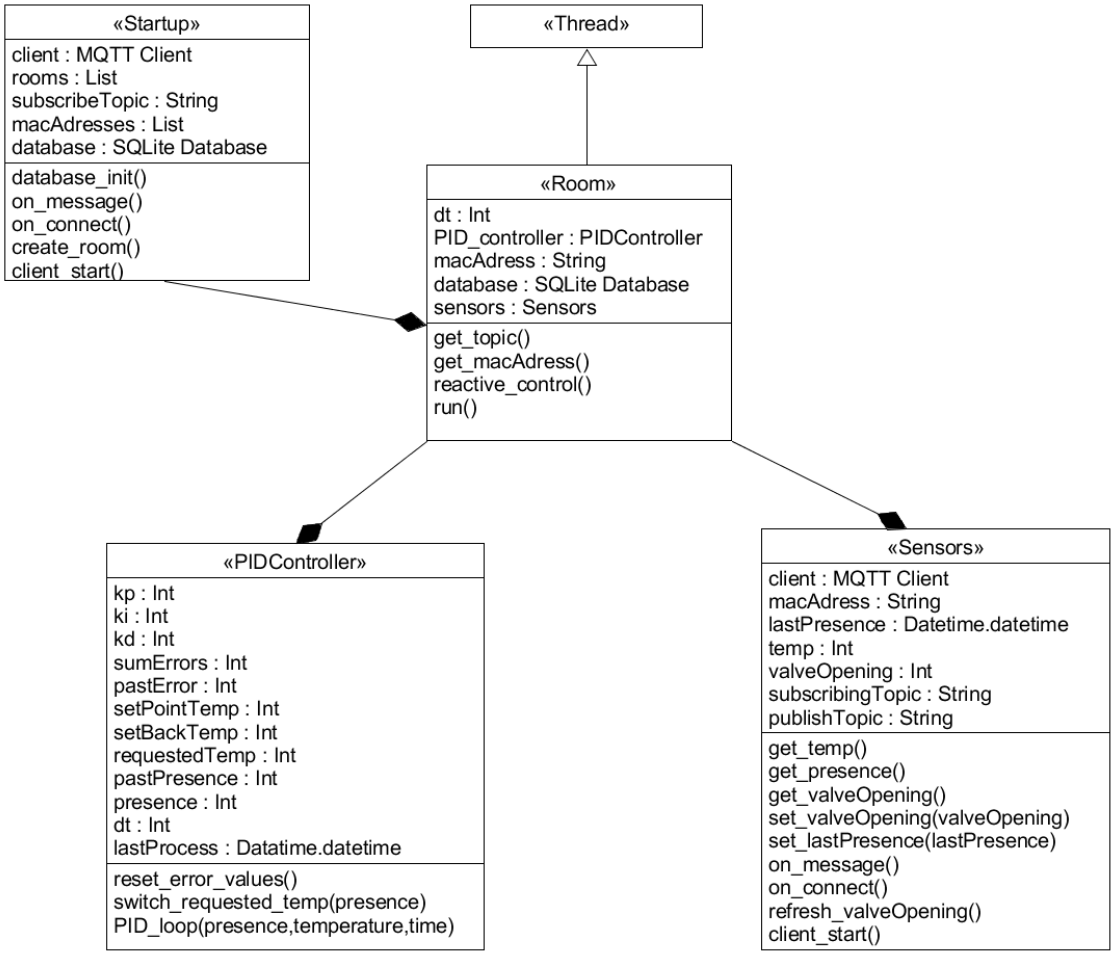


Figure 2 – UML diagram of the Raspberry Pi code

2 Central thermostat

The central thermostat, consisting of a Raspberry Pi (a small and powerful single-board computer), contains the main part of the code. It will process the data coming from thin thermostats in order to control the temperature of the rooms.

To do so, a reactive control algorithm for the temperature has been implemented on the Pi, and is based on a feedback mechanism (represented by the PID Controller class in the UML diagram, Figure 2, page 6).Figure 3 shows how the control algorithm works.

2.1 Temperature control based on a feedback mechanism

At the core of the control algorithm lies its ability to heat each room to a desired temperature, and, eventually, to keep it at said temperature.

Since a temperature sensor was provided as part of the supplied hardware, the sensor was to be used to its full potential.

Hence, a way of utilizing the measured room temperature in order to regulate the thermostat

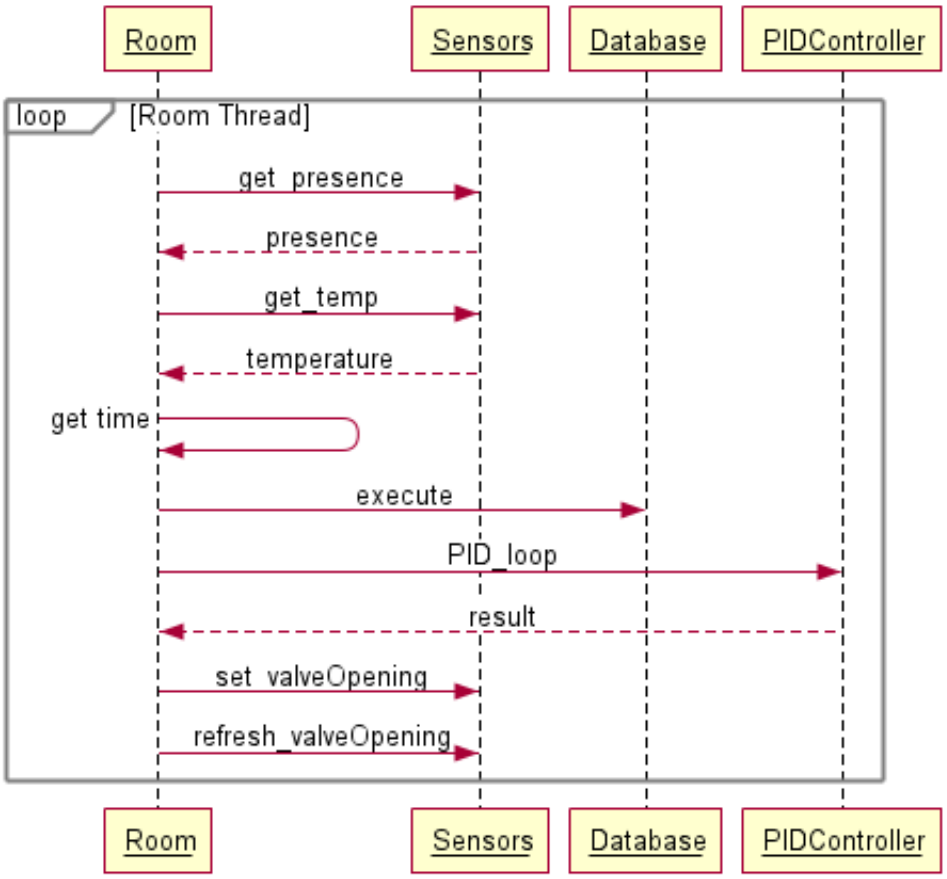


Figure 3 – Sequence diagram of the room thread

output [3] needed to be implemented. This is the fundamental operating principle of feedback control. The general purpose of using feedback control is to bypass the need to mathematically model the studied system [4]. Two variants of feedback control were then looked into : On-Off control and PID control. The following sections describe both control mechanisms, as well as the reasons for choosing the second one.

2.1.1 On-Off control

Suppose a room needs to be at a certain target temperature. If the measured room temperature is below this set-point, the heater is switched on. If, however, it is above the set-point, the heater is switched off. The Table 1 goes over the advantages and disadvantages of On-Off control.

2.1.2 Proportional-Integral-Derivative control

Proportional-Integral-Derivative control, or simply PID control, is a broadly used feedback control mechanism among control engineers. Figure 4 depicts the standard schema of a PID controller. The

3. Valve opening, or in this particular case, a voltage

4. In this case, we avoid having to solve heat equations, as they are outside the scope of this project. This makes the feedback controller easily reusable for rooms with different characteristics

Advantages	Disadvantages
- Ease of implementation	- Constant on and off switching is energy consuming - Significantly reduces the durability of the heat generator - Oscillation of the temperature due to constant on and off switching - Overshoot can not be avoided

Table 1 – Advantages and disadvantages of On-Off control

standard continuous PID equation is the following :

$$u(t) = K_p.e(t) + K_i.\int_0^t e(\tau)\,\mathrm{d}\tau + K_d.\frac{\mathrm{d}e(t)}{\mathrm{d}t} \tag{2.1}$$

where :

- $e(t)$ is the difference between the set-point $r(t)$ and the process variable $y(t)$ (here, the measured room temperature)
- K_p, K_i, and K_d are the PID constants
- $u(t)$ is the command that is given to the actuator (here, the amount by which the valve needs to be opened)

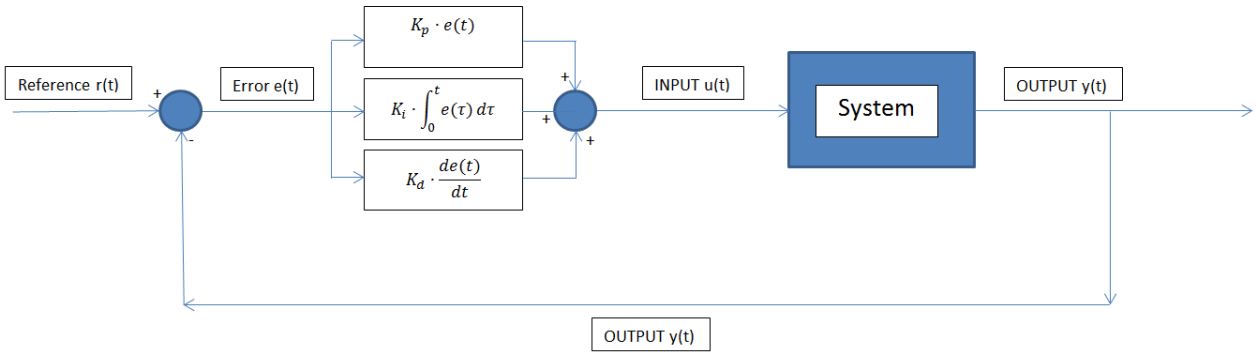


Figure 4 – Standard schema of a PID controller

The thermostat being, in all generality, a computer, the above mentioned equation needs to be discretized in order to be used by said thermostat :

$$u_N = K_p.e_N + K_i.\sum_{i=1}^{N} e_i.\Delta t + K_d.\frac{(e_N - e_{N-1})}{\Delta t} \tag{2.2}$$

where N is the number of iterations done by the control loop up to this point.

The Table 2 explains the perks of using a PID controller.

Advantages	Disadvantages
- Ease of implementation - Effective when tuned properly (significant overshoot and pendeling can be avoided)	- A method for tuning the PID constants automatically is needed

Table 2 – Advantages and disadvantages of PID control

Based on both tables, it was decided to use a PID controller.

2.2 PID tuning

Using a PID controller implies having to tune its constants. They can be tuned two different ways : manually or automatically. The second method goes into the "smart" part of the project, and will be addressed in the second term, as it is still a work in progress.

2.2.1 Manual tuning : a temporary solution

Given that the work on PID control ran in parallel with various other processes, in particular the work on the database, the MQTT protocol and the thin thermostat, a temporary solution for tuning the PID constants was required, as the controller needed to be tested in combination with the remaining of the project.

Manually tuning the constants by doing "field tests" became the first choice, since it allows to bypass the need of mathematically determining the constants. The step-by-step process that was used goes as follows :

1. K_p, K_i, and K_d are all set to 0.

2. The proportional gain K_p is then gradually increased, until the response reaches a compromise between aggressive control and avoiding excessive overshoot. At this point, the P controller should come with a steady-state error. This is because the error $e(t)$ needs to be different from 0 to drive the P controller.

3. The integral gain K_i is then tuned. As the integral term is an accumulation of the error in the past, the goal here is to make the room temperature reach its set-point slightly faster and eliminate the steady-state error that naturally came with the P controller. In fact, suppose

the room temperature is below its set-point, as long as $e(t) \neq 0$, the output of the integral term increases, until the room temperature eventually reaches its set-point.

4. Finally, the derivative gain K_d is gradually increased in order to damp the temperature overshoot and stabilize the system. In fact, the role of the derivative action is to predict system behavior by calculating the slope of the error.

2.2.2 Automatic tuning of the PID constants

While manually tuning the PID constants can be useful at first, as a way of getting used to the range of values you should expect for the coefficients, it can not be used beyond that. In fact, a consumer buying a thermostat should not be expected to know how to tune his thermostat.
Having the PID controller find out the right PID values automatically is a first step towards developing a smart thermostat. Given that this part is still a work in progress, absolute certainty about the way we will be proceeding next term can't be provided. However, two variants of the Ziegler-Nichols method have already been looked into at this point :

- The open loop method (step response method): the goal here is to input a certain value, called the step, into the circuit, and finding out the right constants by analyzing the way the temperature changes over time on a (Temperature, time) graph and using the acquired information to determine the constants.

- The closed loop method (self oscillation method): K_p,K_i and K_d are initially set to zero. The proportional gain is then gradually increased until self-oscillation of the temperature around a certain value. Much like for the first method, the (Temperature, time) graph can then be analyzed in order to determine the right PID values.

2.3 Database Storage

As there is a need for being able to retrieve the previous data, in order for the smart thermostat to plan its heating sessions, it has been decided to store the useful data in a relational and persistent database. A relational database organizes data into tables and is the most widely used type of database.

There are quite a few kinds of databases capable of interacting with Python, but some constraints argue in favor of the SQLite Database. Because the code is running on a Raspberry PI, a priority was for it to be lightweight and run locally.

The Raspberry Pi accesses the database for two purposes. Firstly, it writes the time of storage, day of the week and sensor readings in the table each time the reactive control loop is executed. Secondly, the active learning algorithm will read the data in order to determine when it needs to

preheat a room. The structure of the code being based on multithreading, Python makes sure that only one thread accesses the database at a time, in order to avoid errors coming from simultaneous access to the same piece of information. But for security, a mutex has been programmed but is not used for the moment.

2.4 Smart behavior and machine learning

During the second term, the control algorithm will be improved (by adding functions, or modifying the ones already existing) into a smart algorithm, and the Pi will act as an active learning machine, but will still be based on the same feedback mechanism.

Adapting itself to its environment and its owners' habits, it should be able to:

- minimize the heating energy consumption
- receive commands from the user through a web interface server
- preheat the rooms according to the user's habits
- adapt the inside temperature according to the outside temperature variations
- autotune each room's PID constants (see section 2.2.2, page 10)

3 Thin thermostat

The thin thermostat, consisting of the NodeMcu microcontroller, a pair of sensors and an actuator, is the actual device that will be placed inside the room to be monitored and regulated.

In order to accomplish this task, this thin thermostat sends the data coming from the sensors to the Raspberry Pi and receives the PID command to input into the circuit containing the resistor.

3.1 Hardware

The following components are required for assembling the thin thermostat :

- PIR sensor
- Temperature sensor (DS18B20)
- 4k7 Ohm resistor
- OLED screen
- Breadboard
- NodeMcu

- Jumper Cables

Figure 5 shows the result when all the components are assembled.

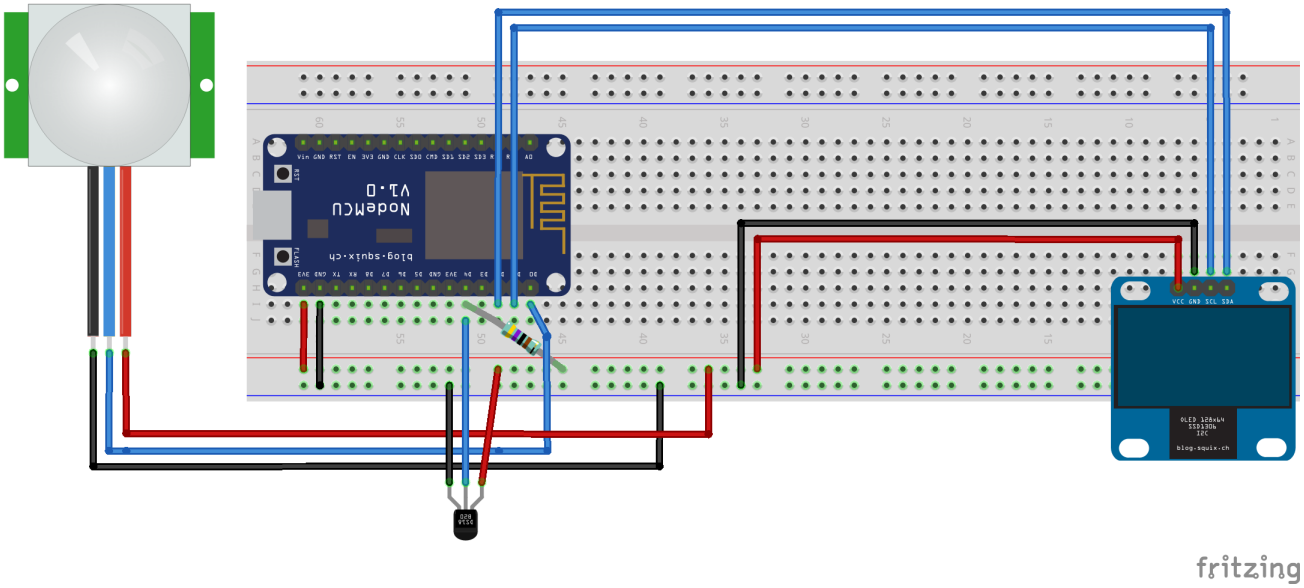


Figure 5 – Wiring diagram of a thin thermostat

The color code used :

- Red Cables - 3.3V (VCC)
- Black Cables - Ground (GND)
- Blue Cables - Data ((OUT) or (SCL and SDA) for screen)

The step-by-step wiring process goes as follows :

1. The NodeMcu needs to be carefully placed in the breadboard. The micro USB port should point towards the exterior of the breadboard, as shown in the diagram, making it easier to plug a power cable later on.
2. A jumper cable is connected to one of the 3.3V ports of the NodeMcu, and to the "+" on one of the sides of the breadboard.
3. Similarly to the step above, a cable connects the GND port to the "-" of the breadboard.
4. All the VCC pins of the components can be connected to a port in the "+" row and all the GND pins to the "-" row.

5. At this point, the 4k7 Ohm resistor can be connected to the D4 port of the NodeMcu and to one port from the "+" row.

6. At last, the data cables need to be connected. The one from the temperature sensor needs to be connected to the D4 port. The PIR sensor is connected to the D0 port. The OLED screen has its SCL port connected to the D1 and the SDA to the D2 port. [5]

3.2 Software

The NodeMcu was coded using a C/C++ type language called the Arduino language.This language shares the same structure with the other two languages, but it doesn't share most of their libraries. However, to make the multiple components and the MQTT protocol work, various specific libraries are needed. Most of the used libraries were provided in the project assignment. The remaining were found on the internet after some research. Only the reason behind the use of these last ones will be justified.

- Adafruit SSD1306 (OLED display library) : though this library is slightly more complicated to use as to the one provided (esp8266-OLED), it has the advantage of allowing to print the content stocked in variables. The provided library only allowed to print plain text, which could only be used for small debugging ; thus, the temperature and the presence state can now easily be printed on the screen, which was impossible to to with the esp8266-OLED

- PubSubclient (MQTT library) : because of the way the the NodeMcu manages its memory allocation for variables, but also the way the topics are handled in this project, the MQTT library needed to be changed, otherwise, the NodeMCU wouldn't be able to subscribe to the correct topic. PubSubClient was therefore chosen, not only because it corrects the issue, but also, because there is a lot of detailed documentation about this particular library, which allows for better understanding

Regarding the temperature sensor, a 11 bit resolution was chosen. This allows for a good compromise between reading speed and temperature precision. In fact, an 11 bit reading will give a temperature with a precision of 0.125 °C. (see Figure 6).

With everything set up, two main functions needed to be coded to make everything work : the "setup" and the "loop". The first one is self explanatory as its job is to start up everything at the NodeMCU boot up. It will connect to the Wi-Fi network using the WIFIManager library and initialize the sensors, the screen and the MQTT (connecting to the broker, setting a callback

5. All the Digital ports given are in accordance with the group's specific code. Other Digital ports could be used, provided that the code is changed accordingly.

Thermometer Resolution Configuration

R1	R0	RESOLUTION (BITS)	MAX CONVERSION TIME	
0	0	9	93.75ms	(t_{CONV}/8)
0	1	10	187.5ms	(t_{CONV}/4)
1	0	11	375ms	(t_{CONV}/2)
1	1	12	750ms	(t_{CONV})

Figure 6 – Different resolutions of the temperature sensor [6]

function and subscribing to a topic [7]). As soon as the setup function ends its job, the loop one takes control of everything, while the NodeMcu stays connected to the power. In order of events, the loop verifies if the NodeMcu client is still connected to the MQTT broker, then it ensures the client is still looping. Next, the loop requests the temperature, prints it on the screen and verifies if there is presence or not. At last, the NodeMcu takes its own MAC Address and the most recent readings and sends them, to their according topic, via the PubSubClient. In the end, the loop delays itself 500ms. This precise time was chose because the Raspberry Pi needs the most recent value every two seconds, so, even the first message is lost, there will be a second message sent before that 2 second. This increases the probability of the Raspberry receiving at least once every two seconds the most recent value.

4 Wireless communication

Building a multi-zone and smart thermostat requires wireless communication in order to overcome the inconvenience of running cables for every single zone equipped with a thin thermostat.

Communication between the thin thermostats and the central unit requires a protocol, which is a standard method of transmitting and processing various kinds of information. There are a variety of protocols for different kinds of information and functions, but the implementation of the MQTT protocol was an assignment of the project.

"The MQTT protocol stands for MQ Telemetry Transport. It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. These principles also turn out to make the protocol ideal of the emerging "machine-to-machine" (M2M) or "Internet of Things" (IoT) world of connected devices." [8]

To facilitate the implementation of this protocol, several libraries were provided. After researching and testing these libraries, the Adafruit MQTT Library was discarded due to some complications and the inability to find a full API documentation. This inconvenience led to choose the PubSub-

7. For more information about how the topics are handled in this project consult Section 4
8. `http://www.mqtt.org`

Client library [9] which provides a client for the thin thermostat enabling this one to connect to an MQTT broker and to subscribe and publish messages. The Eclipse Paho MQTT Python client library [10] provides to the Raspberry Pi an instance of a client and the ability to connect, subscribe and publish.

These clients communicate with each other by means of the broker, which handles all the upcoming messages and sends them to the interested clients. The Mosquitto broker was already pre-installed and running locally on the Raspberry Pi.

Clients publish and subscribe to topics, which are hierarchical structured strings used for filtering and routing messages. These topics are managed by the broker, this means that the broker knows which clients are subscribed (interested) and/or publishing (sending) messages to which topics. Topics are created automatically when a client subscribes/publishes to it.

Each thin thermostat needs its own two topics, one that it will be publishing to, and another one to subscribe to. This creates a problem when there is more than one thin thermostat. It implies that the topic has to include a unique identifier for every thin thermostat. To solve this problem every thin thermostat creates a topic based on its MAC address, it subscribes to **/home/MacAddress/valveAct** and publishes to **/home/MacAddress/readings**, and then it sends its MacAddress to the central thermostat via another topic **/home/start** where the Raspberry Pi is subscribed by default allowing this one to subscribe and publish to the same two topics.

9. `http://pubsubclient.knolleary.net/`
10. `https://pypi.python.org/pypi/paho-mqtt/1.2`

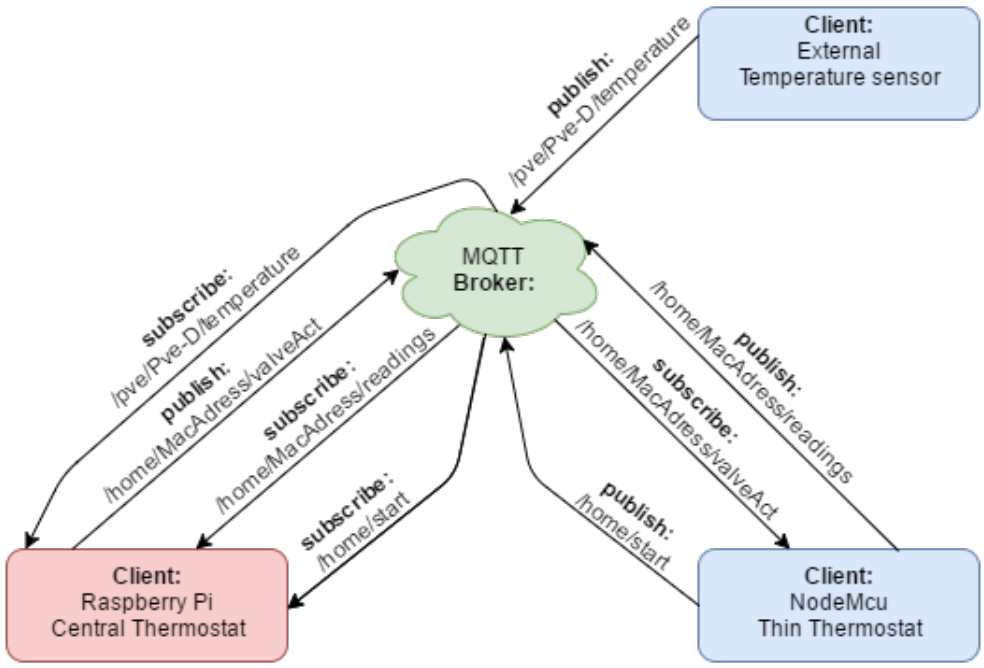


Figure 7 – MQTT diagram

The thin thermostat publishes sensor data ("HIGH" or "LOW" for presence and a certain temperature) to the topic **/home/MacAddress/readings**. This data is received by the central thermostat and gets treated and processed, returning a value which will be published to **/home/MacAddress/valveAct**. This value will be received by the thin thermostat and it will regulate the valve accordingly.

The top right client shown in Figure 7 represents the external temperature sensor, which publishes the outdoor temperature to the **/pve/Pve-D/temperature** topic. This temperature reading is received, treated and processed by the Raspberry Pi.

5 Integration and validation test

With everything set up, preliminary tests were conducted to verify if everything was working properly. First, the NodeMcu needed to be tested in combination with both sensors. To do so, the temperature readings were printed on the OLED Display and compared with a reliable thermostat.However, verifying the working status of the PIR sensor proved to be more complicated. Using MQTT, the presence state was sent to another client that was in another room. That allowed to test presence and non-presence scenarios and read the data while being sure to not disturb the readings from the PIR sensor.

Nonetheless, the biggest test was the verification of the Python code and, in particular, the

analysis of the reactive loop and its use of the PID controller. After ensuring all the code was executing, some tests were conducted to check how the code would handle multiple clients. [11] Those tests were conducted to ensure that the SQLite database could handle multiple threads without the use of a mutex. [12] In order to test the reactive loop and its PID controller, the physical validation environment was used. Seeing that the reactive loop was working as intended , the following step was to manually set up the PID constants [13] to obtain the expected heat curve ; thus, Figure 8, shows how a poorly tuned PID controller behaves. The temperature oscillates a lot and there is significant overshoot (the set point temperature is represented by the black dotted line). This clearly shows one of the weaknesses of manual tuning. It demands a lot of testing and in a lot of cases the result will look like this. In addition to that, manually setting up the constants means that they will only work for the particular environment the tests were concluded in. However, after a lot of tries, it is possible to end up with a reasonable result (Figure 9).Even though there is a slight overshoot followed by a small undershoot, the PID accomplishes its job and finishes by delivering a constant temperature. The goal in the end is to get a similar, if not better, response once the automatic tuning is implemented.

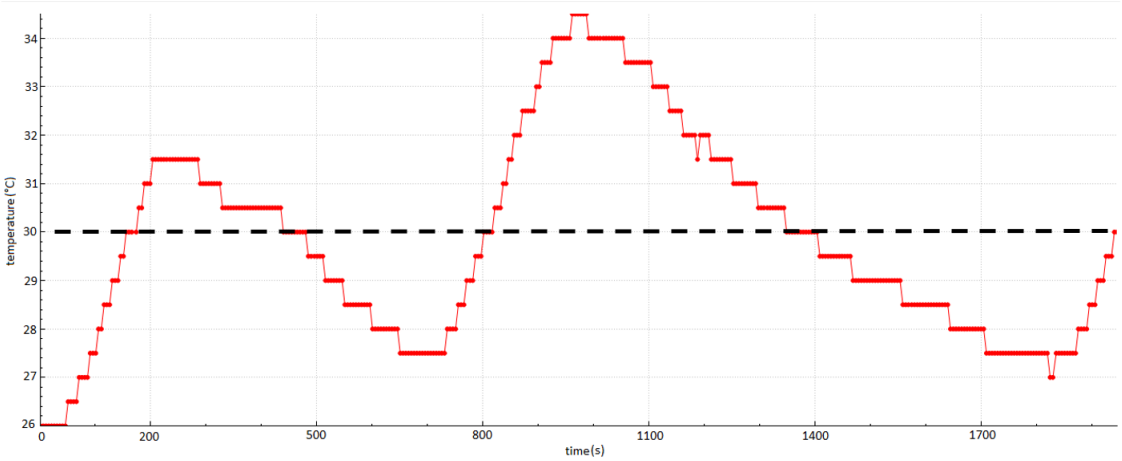


Figure 8 – Heat curve of an incorrectly tuned PID controller ($K_p = 20$, $K_i = 1$, $K_d = 0$)

11. Multiple NodeMcus were simulated creating a small code using Python
12. Even though the database can handle multiple threads without any issues, a mutex is still present in the code but in form of comments. If needed later on its implementation is already done
13. refer to section 2.2.1, Manual tuning : a temporary solution

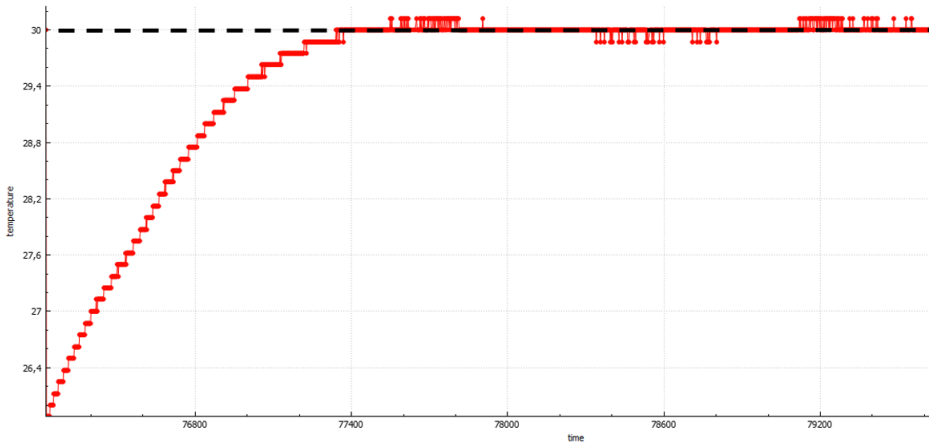


Figure 9 – Heat curve of a correctly tuned PID controller ($K_p = 34$, $K_i = 0.05$, $K_d = 0$)

6 Remaining tasks and conclusion

The second term will consist in developing the smart features of our thermostat. The foundation has been laid, however the fourth milestone is still to be done as a whole. Furthermore, the second half of the third milestone, concerning the web interface, hasn't been worked on as of now. The following smart features have been considered so far :

- Automatic tuning of the PID constants
- Establishment of a mathematical model capable of predicting human presence at any given time.
- Effective preheating of a room, based on this mathematical model, in order to minimize energy consumption
- Interactive web interface
- Taking into account the outside temperature when regulating the inside temperature.

A Group functioning

At the very beginning, a schedule was created to plan our work for the first term (see figure 10, page 19). We informally communicate with each other in a private Facebook group. All the documents and files, in particular the meeting reports, the agenda for the meetings and codes are shared on a Google drive and on a private git. The bibliography report and the intermediary report have been written in LaTeX using the Overleaf platform, in order to allow each member of the team to modify the documents in real time.

The members were split in different groups in order to effectively work in parallel on different parts of the project, in particular the assembling of the thin thermostat circuit, the MQTT protocol for the wireless communication, the establishment of the SQLite database to store the measurements and the reactive control loop. The first subdivision planned was a two groups working, one on the Raspberry and the other on the NodeMCU. Next, the subdivision became more complex as smaller groups were made for working on differents parts of the code (database, wireless communication, PID, Control algorithm and NodeMCU main body) and merging them together using a private Git called GitHub. This last subdivision seemed to be the best because it's closer to the way the structure of the code was seen.

The meetings between the students then focused on giving everybody the opportunity to share their work with the other members of the group, and combining all the pieces of codes in order to test the program as a whole. Each week we met with our tutor in order to keep him abreast of the advances made, while one leader and one secretary were chosen for each meeting.

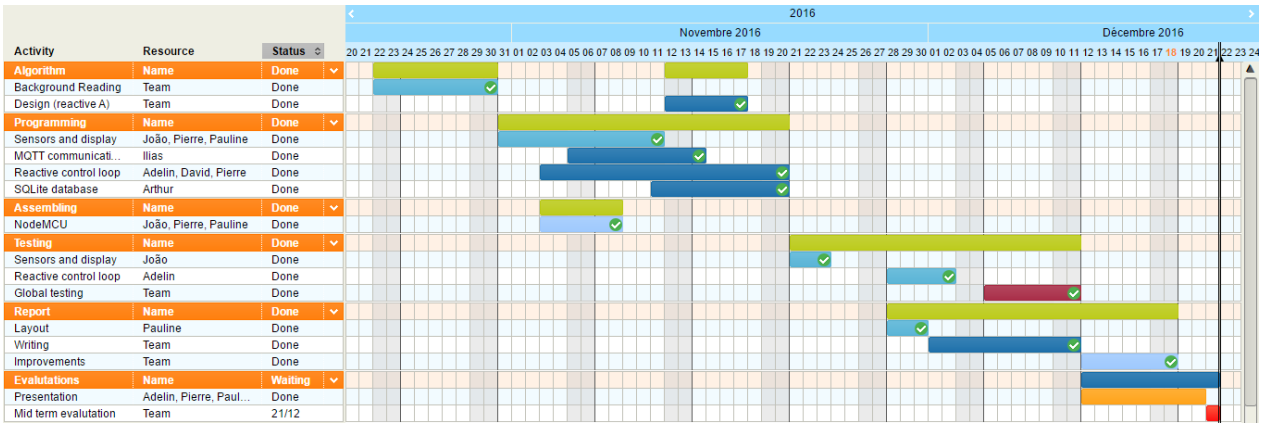
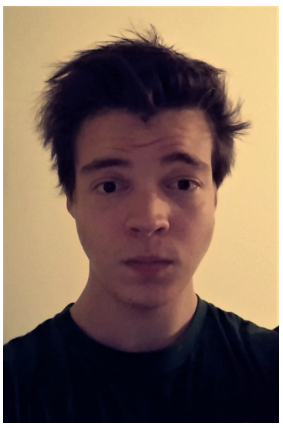


Figure 10 – Planning Gantt for the first semester

B Group members



(a) Maoujoud David, 427921, dmaoujou@ulb.ac.be

(b) Marques Correia João, 408721, jmarques@ulb.ac.be

(c) Pereira Acuna Pierre, 408630, ppereira@ulb.ac.be

(d) Roty Adelin, 428286, aroty@ulb.ac.be

(e) Serroukh Mardi Ilias, 408644, iserrouk@ulb.ac.be

(f) Van Heirstraeten Arthur, 408667, avheirst@ulb.ac.be

(g) Willaert Pauline, 408676, pwillaer@ulb.ac.be

Figure 11 – Group members

References

[1] Adafruit_ssd1306. URL https://github.com/adafruit/Adafruit_SSD1306.

[2] Arduino - software. URL https://www.arduino.cc/en/Main/Software.

[3] Convert/reading payload to int, float, and string / better solution? · issue #105 · knolleary/pubsubclient. URL https://github.com/knolleary/pubsubclient/issues/105.

[4] Datatypes in SQLite version 3. URL https://www.sqlite.org/datatype3.html.

[5] Database Programming - Python Wiki. URL https://wiki.python.org/moin/DatabaseProgramming.

[6] Ds18b20 data sheet. URL http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf.

[7] Esp8266 WIFI/Arduino. URL https://github.com/esp8266/Arduino.

[8] klarsys/esp8266-OLED. URL https://github.com/klarsys/esp8266-OLED.

[9] Get module's MAC address · issue #313 · esp8266/arduino. URL https://github.com/esp8266/Arduino/issues/313.

[10] Messaging reliability and persistence with the MQTT protocol. URL http://thenewstack.io/messaging-reliability-persistence-mqtt/.

[11] MQTT version 3.1.1. URL http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

[12] Cours "première approche des classes" sur @OpenClassrooms. URL https://openclassrooms.com/courses/apprenez-a-programmer-en-python/premiere-approche-des-classes.

[13] Arduino client for MQTT. URL http://pubsubclient.knolleary.net/.

[14] Web Frameworks - Full Stack Python, . URL https://www.fullstackpython.com/web-frameworks.html.

[15] Web Frameworks - Python Wiki, . URL https://wiki.python.org/moin/WebFrameworks.

[16] Hong Linh Truong Andy Stanford-Clark. MQTT for sensor networks (MQTT-SN) protocol specification. URL http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf.

[17] J. Crowe and M. A. Johnson. On-line model-free methods. In Michael A. Johnson and Mohammad H. Moradi, editors, *PID Control*, pages 109–146. Springer London, . ISBN

978-1-85233-702-5 978-1-84628-148-8. URL http://link.springer.com/chapter/10.1007/1-84628-148-2_3. DOI: 10.1007/1-84628-148-2_3.

[18] J. Crowe and M. A. Johnson. Phase-locked loop methods and PID control. In Michael A. Johnson and Mohammad H. Moradi, editors, *PID Control*, pages 259–296. Springer London, . ISBN 978-1-85233-702-5 978-1-84628-148-8. URL http://link.springer.com/chapter/10.1007/1-84628-148-2_7. DOI: 10.1007/1-84628-148-2_7.

[19] J. Crowe and M. A. Johnson. Phase-locked loop methods. In Michael A. Johnson and Mohammad H. Moradi, editors, *PID Control*, pages 213–257. Springer London, . ISBN 978-1-85233-702-5 978-1-84628-148-8. URL http://link.springer.com/chapter/10.1007/1-84628-148-2_6. DOI: 10.1007/1-84628-148-2_6.

[20] Anna Calveras Ernesto García Davis and Ilker Demirkol. Improving packet delivery performance of publish/subscribe protocols in wireless sensor networks. URL http://www.mdpi.com/1424-8220/13/1/648/htm.

[21] Zhili Wang Fangling Pu, Wenchao Zhang Chong Du, and Nengcheng Chen. Semantic integration of wireless sensor networks into open geospatial consortium sensor observation service to access and share environmental monitoring systems. 10(2):p. 45 – 53. ISSN 1751-8814. URL http://digital-library.theiet.org.ezproxy.ulb.ac.be/content/journals/10.1049/iet-sen.2014.0141.

[22] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback control of dynamic systems*. Upper Saddle River, NJ : Prentice Hall, c2002., 4th ed. edition. ISBN 0-13-032393-4. URL http://cibleplus.ulb.ac.be.ezproxy.ulb.ac.be/recorddetails/757607?0#.

[23] John Goerzen and Brandon Rhodes. The world wide web. In *Foundations of Python Network Programming*, pages 183–222. Apress. ISBN 978-1-4302-5854-4 978-1-4302-5855-1. URL http://link.springer.com/chapter/10.1007/978-1-4302-5855-1_11. DOI: 10.1007/978-1-4302-5855-1_11.

[24] Magnus Lie Hetland. Batteries included. In *Beginning Python*, pages 209–260. Apress, . ISBN 978-1-59059-982-2 978-1-4302-0634-7. URL http://link.springer.com/chapter/10.1007/978-1-4302-0634-7_10. DOI: 10.1007/978-1-4302-0634-7_10.

[25] Magnus Lie Hetland. Database support. In *Beginning Python*, pages 293–304. Apress, . ISBN 978-1-59059-982-2 978-1-4302-0634-7. URL http://link.springer.com/chapter/10.1007/978-1-4302-0634-7_13. DOI: 10.1007/978-1-4302-0634-7_13.

[26] Magnus Lie Hetland. Files and stuff. In *Beginning Python*, pages 261–275. Apress, . ISBN 978-1-59059-982-2 978-1-4302-0634-7. URL http://link.springer.com/chapter/10.1007/978-1-4302-0634-7_11. DOI: 10.1007/978-1-4302-0634-7_11.

[27] H.-P. Huang and J.-C. Jeng. Process reaction curve and relay methods identification and PID tuning. In Michael A. Johnson and Mohammad H. Moradi, editors, *PID Control*, pages 297–337. Springer London. ISBN 978-1-85233-702-5 978-1-84628-148-8. URL http://link.springer.com/chapter/10.1007/1-84628-148-2_8. DOI: 10.1007/1-84628-148-2_8.

[28] M. A. Johnson. PID control technology. In Michael A. Johnson and Mohammad H. Moradi, editors, *PID Control*, pages 1–46. Springer London. ISBN 978-1-85233-702-5 978-1-84628-148-8. URL http://link.springer.com/chapter/10.1007/1-84628-148-2_1. DOI: 10.1007/1-84628-148-2_1.

[29] M. A. Johnson and M. H. Moradi. Some PID control fundamentals. In Michael A. Johnson and Mohammad H. Moradi, editors, *PID Control*, pages 47–107. Springer London. ISBN 978-1-85233-702-5 978-1-84628-148-8. URL http://link.springer.com/chapter/10.1007/1-84628-148-2_2. DOI: 10.1007/1-84628-148-2_2.

[30] Michel Kinnaert. Automatique MATH - h304.

[31] P. Martin, M. J. Grimble, D. Greenwood, and M. A. Johnson. Restricted structure optimal control. In Michael A. Johnson and Mohammad H. Moradi, editors, *PID Control*, pages 429–472. Springer London. ISBN 978-1-85233-702-5 978-1-84628-148-8. URL http://link.springer.com/chapter/10.1007/1-84628-148-2_12. DOI: 10.1007/1-84628-148-2_12.

[32] Michael McRoberts. Temperature sensors. In *Beginning Arduino*, pages 271–284. Apress. ISBN 978-1-4302-5016-6 978-1-4302-5017-3. URL http://link.springer.com/chapter/10.1007/978-1-4302-5017-3_13. DOI: 10.1007/978-1-4302-5017-3_13.

[33] M. H. Moradi and M. R. Katebi. Relay experiments for multivariable systems. In Michael A. Johnson and Mohammad H. Moradi, editors, *PID Control*, pages 183–211. Springer London. ISBN 978-1-85233-702-5 978-1-84628-148-8. URL http://link.springer.com/chapter/10.1007/1-84628-148-2_5. DOI: 10.1007/1-84628-148-2_5.

[34] M. H. Moradi, M. A. Johnson, and M. R. Katebi. Predictive PID control. In Michael A. Johnson and Mohammad H. Moradi, editors, *PID Control*, pages 473–529. Springer London. ISBN 978-1-85233-702-5 978-1-84628-148-8. URL http://link.springer.com/chapter/10.1007/1-84628-148-2_13. DOI: 10.1007/1-84628-148-2_13.

[35] Sebastian Raschka. *Python Machine Learning*. Packt Publishing Ltd. ISBN 978-1-78355-514-7. Google-Books-ID: GOVOCwAAQBAJ.

[36] Brandon Rhodes and John Goerzen. Caches and message queues. In *Foundations of Python Network Programming*, pages 137–150. Apress, . ISBN 978-1-4302-5854-4 978-1-4302-5855-1. URL http://link.springer.com/chapter/10.1007/978-1-4302-5855-1_8. DOI: 10.1007/978-1-4302-5855-1_8.

[37] Brandon Rhodes and John Goerzen. HTTP clients. In *Foundations of Python Network Programming*, pages 151–168. Apress, . ISBN 978-1-4302-5854-4 978-1-4302-5855-1. URL http://link.springer.com/chapter/10.1007/978-1-4302-5855-1_9. DOI: 10.1007/978-1-4302-5855-1_9.

[38] Brandon Rhodes and John Goerzen. HTTP servers. In *Foundations of Python Network Programming*, pages 169–182. Apress, . ISBN 978-1-4302-5854-4 978-1-4302-5855-1. URL http://link.springer.com/chapter/10.1007/978-1-4302-5855-1_10. DOI: 10.1007/978-1-4302-5855-1_10.

[39] Brandon Rhodes and John Goerzen. Introduction to client-server networking. In *Foundations of Python Network Programming*, pages 1–16. Apress, . ISBN 978-1-4302-5854-4 978-1-4302-5855-1. URL http://link.springer.com/chapter/10.1007/978-1-4302-5855-1_1. DOI: 10.1007/978-1-4302-5855-1_1.

[40] Brandon Rhodes and John Goerzen. Network data and network errors. In *Foundations of Python Network Programming*, pages 75–92. Apress, . ISBN 978-1-4302-5854-4 978-1-4302-5855-1. URL http://link.springer.com/chapter/10.1007/978-1-4302-5855-1_5. DOI: 10.1007/978-1-4302-5855-1_5.

[41] Brandon Rhodes and John Goerzen. Server architecture. In *Foundations of Python Network Programming*, pages 115–136. Apress, . ISBN 978-1-4302-5854-4 978-1-4302-5855-1. URL http://link.springer.com/chapter/10.1007/978-1-4302-5855-1_7. DOI: 10.1007/978-1-4302-5855-1_7.

[42] Brandon Rhodes and John Goerzen. Socket names and DNS. In *Foundations of Python Network Programming*, pages 57–74. Apress, . ISBN 978-1-4302-5854-4 978-1-4302-5855-1. URL http://link.springer.com/chapter/10.1007/978-1-4302-5855-1_4. DOI: 10.1007/978-1-4302-5855-1_4.

[43] A. Sanchez, M. R. Katebi, and M. A. Johnson. Tuning PID controllers using subspace identification methods. In Michael A. Johnson and Mohammad H. Moradi, editors, *PID Control*, pages 361–388. Springer London. ISBN 978-1-85233-702-5 978-1-84628-148-8. URL

http://link.springer.com/chapter/10.1007/1-84628-148-2_10. DOI: 10.1007/1-84628-148-2_10.

[44] K. K. Tan, T. H. Lee, and R. Ferdous. Automatic PID controller tuning — the nonparametric approach. In Michael A. Johnson and Mohammad H. Moradi, editors, *PID Control*, pages 147–181. Springer London. ISBN 978-1-85233-702-5 978-1-84628-148-8. URL http://link.springer.com/chapter/10.1007/1-84628-148-2_4. DOI: 10.1007/1-84628-148-2_4.

[45] K. S. Tang, G. R. Chen, K. F. Man, and S. Kwong. Fuzzy logic and genetic algorithm methods in PID tuning. In Michael A. Johnson and Mohammad H. Moradi, editors, *PID Control*, pages 339–360. Springer London. ISBN 978-1-85233-702-5 978-1-84628-148-8. URL http://link.springer.com/chapter/10.1007/1-84628-148-2_9. DOI: 10.1007/1-84628-148-2_9.

[46] Weng Tat Leong Valerie Lampkin, Sweta Rawat Leonardo Olivera, and Rong Xiang Nagesh Subrahmanyam. *Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry*. ISBN 978-0-7384-3708-8. URL http://www.redbooks.ibm.com/abstracts/sg248054.html.

[47] Vance VanDoren. PID–the basic technique for feedback control. 44(1):132. ISSN 00108049. URL http://search.proquest.com.ezproxy.ulb.ac.be/docview/200411058/abstract/8DAD19148F34C5DPQ/1.

[48] Antonio Visioli. *Practical PID Control*. Advances in Industrial Control. Springer London. ISBN 978-1-84628-585-1. URL http://link.springer.com/10.1007/1-84628-586-0.

[49] Q.-G. Wang, Yong Zhang, and Yu Zhang. Design of multi-loop and multivariable PID controllers. In Michael A. Johnson and Mohammad H. Moradi, editors, *PID Control*, pages 389–427. Springer London. ISBN 978-1-85233-702-5 978-1-84628-148-8. URL http://link.springer.com/chapter/10.1007/1-84628-148-2_11. DOI: 10.1007/1-84628-148-2_11.