ECOLE POLYTECHNIQUE DE BRUXELLES

ULB UNIVERSITÉ LIBRE DE BRUXELLES

Rapport final

TRAN-H-201

« Le Thermostat Intelligent »

Étudiants :

Amzur Soufiane
Daubry Wilson
Sercu Stéphane
Verstraeten Denis

Superviseur :

Eppe Stefan

Lecteur :

Vermeir Aurélien

14 Mars 2016

Abstract

"Realisation of a Smart Thermostat using Raspberry Pi and Arduino Nano", by Amzur Soufiane, Daubry Wilson, Sercu Stéphane and Verstraeten Denis, Université Libre de Bruxelles, 2015-2016.

The purpose of this project is to design a smart thermostat. The principle is simple : a Raspberry Pi 2 Model B with Python and a Wifi antenna communicates with « thin thermostats » thanks to the http protocol. These « thin thermostats » are microcontrollers supplied with a thermistor, a presence sensor and a module Wifi. Data collected can be gathered and stored in a SQLite database. As a first step, a simple reactive control has to be implemented in the central server. It consists of to open or to close thermostatic valve depending on whether it is too cold or too hot. A smarter and more predictive control algorithm will be then implemented using databases in order to learn the habits and the preferences of users. At the end, it will be necessary to set up a rigourous test protocol. The aim of those tests is to ensure the quality of the system as well quantitatively as qualitatively.

Key words : thermostat, microcontroller, smart, databases, server, data mining, machine learning
[Word limit : 160]

"Realisation d'un thermostat intelligent avec un Arduino et un Raspberry Pi", par Amzur Soufiane, Daubry Wilson, Sercu Stéphane et Verstraeten Denis, Université Libre de Bruxelles, 2015-2016.

Le but de ce projet est de créer un système de thermostats intelligents. Un thermostat central doit être implémenté en Python sur un Raspberry Pi 2B équipé d'une antenne Wi-Fi avec laquelle il communique suivant le protocole HTTP avec les « thin thermostats » . Ces derniers sont composés d'un micro-contrôleur Arduino Nano V2, d'un capteur de présence, d'une thermistance et d'un module Wi-Fi, le tout monté sur une BreadBord. Un environnement y est également connecté, lequel possède une vanne de chauffage et alimente l'Arduino. Ils servent uniquement à prendre des mesures stockées par le Rapsberry dans une base de données implémentée en SQLite et à actualiser la valeur de la vanne. Un algorithme de contrôle réactif est premièrement intégré, suivant le principe du « ON/OFF ». Il est suivi d'un algorithme intelligent et prédictif se basant sur le « vécu » du thermostat et les habitudes de l'utilisateur, il se veut également écologique en minimisant les oscillations de température à l'aide d'un régulateur PID. Des stratégies de tests et une validation du modèle devront être mis en place dans l'objectif de dégager des indicateurs objectifs de performance.

Mots-clés : thermostat, microcontrôleur, intelligence, base de données, serveur
[Limite de mots : 188]

Table des matières

Table des figures

Liste des tableaux

1 Introduction

Actuellement, il y a une très forte augmentation de l'offre et de la demande d'objets connectés ou intelligents, car le désir d'une utilisation simplifiée et automatisée est grandissant. C'est dans cette optique que s'inscrit ce projet multidisciplinaire orienté vers l'informatique : « Thermostat intelligent ». En effet, le dispositif à concevoir se doit d'être autonome, doté d'un modèle de prédiction personnel, adapté aux habitudes de l'utilisateur et minimisant la consommation énergétique en évitant les oscillations de température.

Le projet d'année des étudiants en bloc deux de l'école polytechnique consiste en la réalisation d'un thermostat intelligent. Ce dernier sera composé de deux types de périphériques, une unité centrale qui sera un Raspberry Pi 2 et des « Remote Devices » qui seront des Arduinos Nano.

Dans ce rapport, la conception globale du thermostat est abordée en premier. L'implémentation en language Arduino [1] (proche du C) et son câblage suit, avant de passer au thermostat central. Le développement des outils indispensables à son fonctionnement, tels qu'un serveur, une base de données, un processus de récolte de données et un régulateur en boucle fermée est d'abord abordée. Ensuite les différent modèles prédictifs, se basant sur les précédents outils et servant l'intelligence du thermostat ainsi que leur implémentation sont présentés. L'intelligence du code en elle même suivra. Cette partie étant composée de la prédiction de présence et du temps de chauffe. Une validation de ces modèles afin vérifier que ces dispositifs sont fonctionnels et efficaces est ensuite explicitée, avant de poursuivre avec le fonctionnement de groupe. Cette partie détaille la manière dont le travail à été réparti et organisé. Des perspectives d'évolutions sont également envisagées, celles-ci listant principalement des fonctionnalités et des amélioration qu'il serait intéressant d'ajouter au prototype.

Les notations propres au langage UML 2.0 sont utilisées. Les conventions d'association, d'agrégation, d'héritage, de public, privé sont conservées. Les schémas de comportement et d'initialisation, correspondent aux diagrammes d'interaction du langage. Dans ces derniers, les cadres ne respectent pas le formalisme usité. Les formes des cadres principalement, ces diagrammes ayant pour but principal d'assurer a compréhension du fonctionnement du thermostat, ils ne se veulent pas extrêmement rigoureux.

Il est également important de mentionner que ce rapport ne fait pas état des codes exacts qui se trouvent dans le dépôt GIT, il expose la structure envisagée. En effet, suite à des problèmes liés au module Wi-Fi de l'Arduino indépendants de notre volonté, la cadence de travail dû être fortement ralentie. Durant plusieurs semaines, la communication entre le thermostat central et les Remotes Devices fut impossible. Les tests ne furent plus possibles, les vérifications de l'avancement du travail non plus. C'est donc pour cela que certains tests ne sont pas concluant. Des alternatives durent donc être trouvées, mais il ne fut pas possible d'en déterminer pour tout.

1. Sketch

2 Conception

Afin de commencer sur une bonne base et d'offrir une vue globale du projet, un diagramme (présenté en Annexe A) a été construit. Le but premier de celui-ci est d'offrir une vue globale et commune du développement à tous ses contributeurs. Cela permet principalement le partage efficace des différentes parties à développer entre les participants. Le rassemblement de tous les modules est également grandement simplifié. En effet, il est beaucoup plus facile d'assembler des parties du projet qui ont été construites en gardant en tête la place qu'elles occupent dans son ensemble et la façon dont elles sont sensées communiquer les unes avec les autres.

En plus de tous ces avantages pratiques, ce diagramme a un grand pouvoir documentaire puisqu'il offre une vision synthétique et schématique de l'ensemble du projet. Il n'a pas été construit en une fois. En effet, il a été complété et corrigé lors de l'avancée du développement en fonction des discussions menées concernant certains problèmes et choix qui ont dû être opérés. Le diagramme est explicité et décrit plus en détails dans le reste du rapport puisque chacune des parties principales du projet est détaillée sur cette base.

3 Thermostats auxiliaires

1 Hardware

D'un point de vue hardware, le montage de l'Arduino Nano est réalisé sur une BreadBord. Les éléments ayant été intégrés à ce câblage sont :

— Un Arduino Nano V2 [3]

— Une thermistance et une résistance de 10kΩ [4]

— Un capteur de présence infrarouge (PIR [1] sensor) [8]

— Un module Wi-Fi ESP-8266 [35]

— Une vanne thermostatique

Ces quatre composants doivent être alimentés et leur signaux de retour captés par le microcontrôleur. Leurs spécificités d'utilisation indiquées dans leur datasheet respective sont prises en compte afin d'éviter que le matériel ne soit endommagé. Ils sont donc branchés suivant la logique suivante :

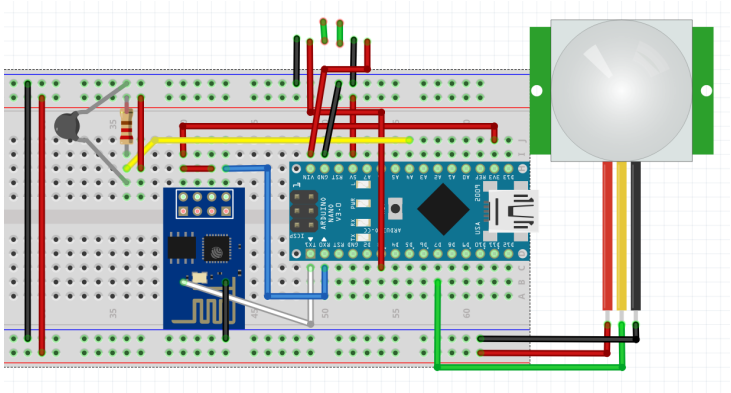


FIGURE 3.1 – Câblage du microcontrôleur

Le détecteur de présence est muni de 3 câbles, un servant d'alimentation permettant de recevoir la tension de 5V, un autre étant la terre du circuit et le dernier devant être relié à une pin numérique afin de lire les données fournies par le détecteur. La pin numérique choisie est la pin D7. Ce choix est purement arbitraire, la seule obligation étant que l'entrée soit digitale.

La thermistance nécessite également une tension d'entrée de 5V, celle-ci doit au préalable passer dans une résistance de 10kΩ branchée en série afin que la Loi d'Ohm puisse lui être appliquée. Elle est finalement reliée à la masse. Une lecture analogique du capteur est effectuée au

1. Passive Infrared

moyen d'un câble reliant la pin A4 à la tension qui entre dans la thermistance.

Le branchement de module Wi-Fi est un petit peu plus complexe. D'abord, contrairement aux autres composants, il est connecté à une source d'alimentation de 3.3V délivrée par le microcontrôleur. Cette tension doit être dirigée vers les pins « VCC » et « CH_PD » du module Wi-Fi. Comme tous les autres composants, sa borne « Ground » est connectée à la masse. Notons enfin la présence de 2 câbles indispensables qui assurent la communication entre l'Arduino et le module Wi-Fi, ce sont les câbles qui relient respectivement les bornes « Rx » et « Tx » du premier aux bornes « Tx » et « Rx » du second.

Enfin, il est à noter que l'Arduino est alimenté par un environnement physique grâce aux pin V_{in} et *Gnd*. Ce premier « renvoie » cette alimentation au travers de sa sortie 5V. L'Arduino peut également transmettre des ordres d'ouverture ou de fermeture partielle de la vanne au travers de la pin D3 directement reliée à l'environnement.

Les six fils isolés donnant l'impression de n'être branchés à rien sont en fait connectés à l'environnement. De droite à gauche se situent la masse de la vanne, son actuateur, deux câbles symbolisant une thermistance présente sur l'environnement, la masse de l'Arduino et son alimentation.

2 Software

D'un point de vue du software, il faut que le microcontrôleur soit capable de lire les informations des différents capteurs mais qu'il puisse aussi générer à partir de cette lecture un type d'information exploitable. Une structure de base de code a été fournie. Elle est axée sur une optimisation de la mémoire, assez réduite, de l'Arduino et elle est complétée par les fonctions nécessaires.

Le PIR n'envoie que deux types de signaux, à savoir présence ou pas de présence. Ceux-ci auront la forme de HIGH ou LOW, respectivement. Ils seront convertis en booléens afin d'être plus facilement manipulables par le serveur.

Quant à la valve thermostatique, une variable globale contenant la valeur de son ouverture est stockée dans le code. Dès lors, lorsque qu'un ordre d'actualisation est reçu, la tension correspondante est envoyée et la valeur de la variable est mise à jour. Lorsqu'on souhaite consulter la valeur de l'ouverture, la variable est renvoyée. De cette façon, il est possible d'effectuer une lecture sans nécessiter de câblages supplémentaires.

Pour la mesure de la température, le calcul est un peu plus complexe. D'abord, ce qui sort du senseur est une valeur analogique comprise entre 0 et 1023 qui représente la différence de potentiel aux bornes de la thermistance. Elle dépend bien entendu de la température du milieu. Dans l'état actuel des choses, une valeur de 5 V est utilisée dans la relation de conversion. Il a toutefois été envisagé d'effectuer une lecture analogique de cette tension et d'injecter la valeur trouvée dans l'expression afin qu'aucun biais sur la température n'apparaisse, ceci rendant le dispositif indépendant de la qualité de la source électrique qui l'alimente [2].

Une fois la tension aux bornes de la thermistance connue, on peut déterminer sa résistance en effectuant un simple calcul de diviseur de tension. C'est en utilisant cette valeur qu'il est possible de déterminer la température de l'environnement grâce à la relation Steinhart-Hart[4] :

2. Par exemple, lorsque l'Arduino est alimenté par un port USB, il ne délivre plus que 4.64 V.

$$T(R) = (A + B\ \ln(\frac{R}{Ref}) + C\ \ln^2(\frac{R}{Ref}) + D\ \ln^3(\frac{R}{Ref}))^{-1} \tag{3.1}$$

Les constantes de cette relation sont déterminées grâce à la datasheet de la thermistance (voir Table 3.1 [4]). Tous les détails correspondants aux calculs se trouvent dans le code.

PARAMETER FOR DETERMINING NOMINAL RESISTANCE VALUES												
NUMBER	$B_{25/85}$ (K)	NAME	TOL. B (%)	A	B (K)	C (K^2)	D (K^3)	A_1	B_1 (K^{-1})	C_1 (K^{-2})	D_1 (K^{-3})
1	2880	Mat O. with Bn = 2880K	3	- 9.094	2251.74	229098	- 2.744820E+07	3.354016E-03	3.495020E-04	2.095959E-06	4.260615E-07
8	3740	Mat B. with Bn = 3740K	2	- 13.8973	4557.725	- 98275	- 7.522357E+06	3.354016E-03	2.744032E-04	3.666944E-06	1.375492E-07
9	3977	Mat A. with Bn =3977K	0.75	- 14.6337	4791.842	- 115334	- 3.730535E+06	3.354016E-03	2.569850E-04	2.620131E-06	6.383091E-08
10	4090	Mat C. with Bn = 4090K	1.5	- 15.5322	5229.973	- 160451	- 5.414091E+06	3.354016E-03	2.519107E-04	3.510939E-06	1.105179E-07
11	4190	Mat D. with Bn = 4190K	1.5	- 16.0349	5459.339	- 191141	- 3.328322E+06	3.354016E-03	2.460382E-04	3.405377E-06	1.034240E-07
12	4370	Mat E. with Bn = 4370K	2.5	- 16.8717	5759.15	- 194267	- 6.869149E+06	3.354016E-03	2.367720E-04	3.585140E-06	1.255349E-07
13	4570	Mat F. with Bn = 4570K	1.5	- 17.6439	6022.726	- 203157	- 7.183526E+06	3.354016E-03	2.264097E-04	3.278184E-06	1.097628E-07

TABLE 3.1 – Tableau des constantes utilisées de la datasheet du thermistor

Les Figures 3.2 et 3.3 illustrent l'implémentation du microcontrôleur. On y retrouve les deux étapes par lesquelles passe l'Arduino : le `setup` et le `loop`.

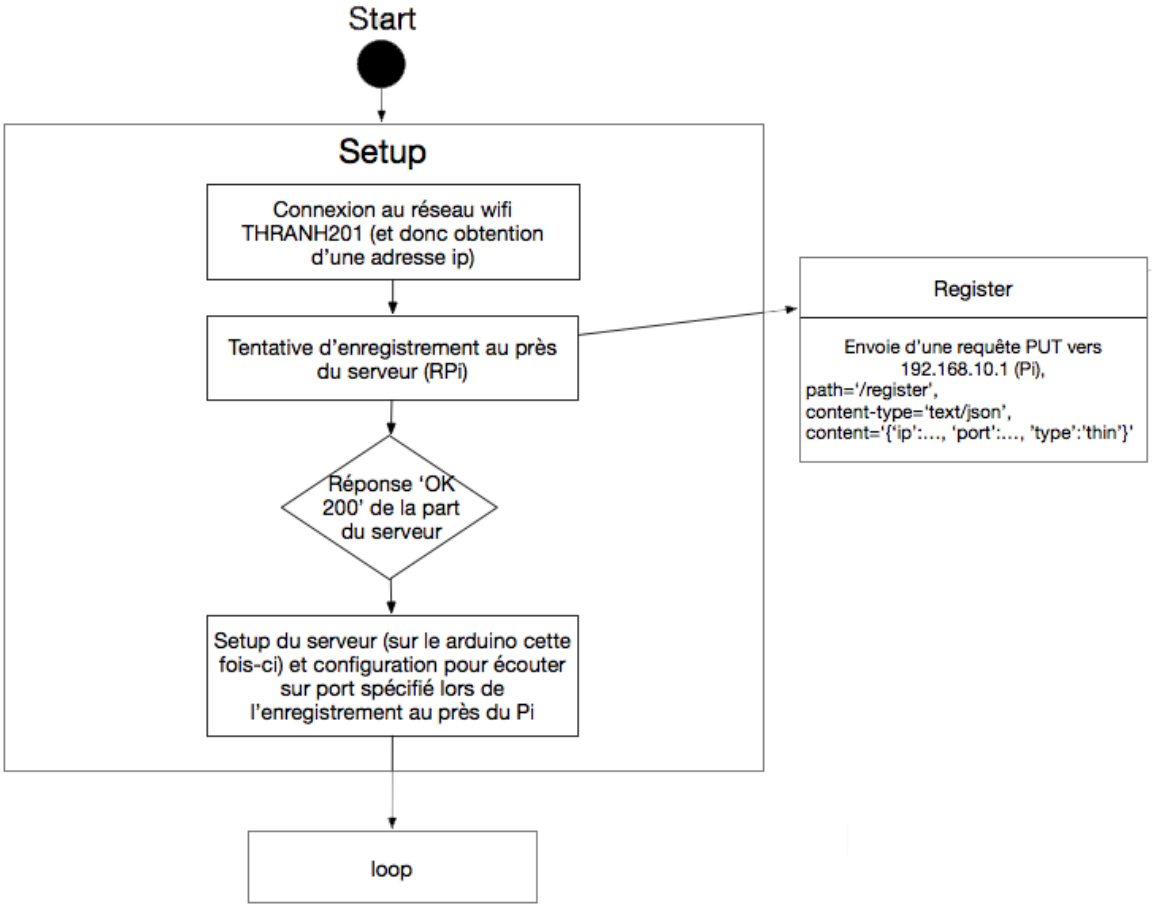


FIGURE 3.2 – Schéma des étapes d'initialisation d'un arduino

Le `setup` se lance à la mise sous tension de l'Arduino et est effectué une seule fois. Il est donc logique d'y retrouver la fonction d'initialisation de connexion au Wi-Fi du serveur et celle

d'enregistrement. La valve thermostatique est également initialisée à 0.

Le `loop`, quant à lui, est la boucle qui tourne tant que l'Arduino est sous-tension. Il vérifie à intervalle régulier qu'aucune requête ne lui est adressée. Dans le cas échéant, il découpe la requête et compose sa réponse en fonction de celle-ci. Si elle est bien adressée, alors il renvoie l'information demandée en respectant le protocole HTTP. Sinon, il envoie la réponse « 400 » au serveur.

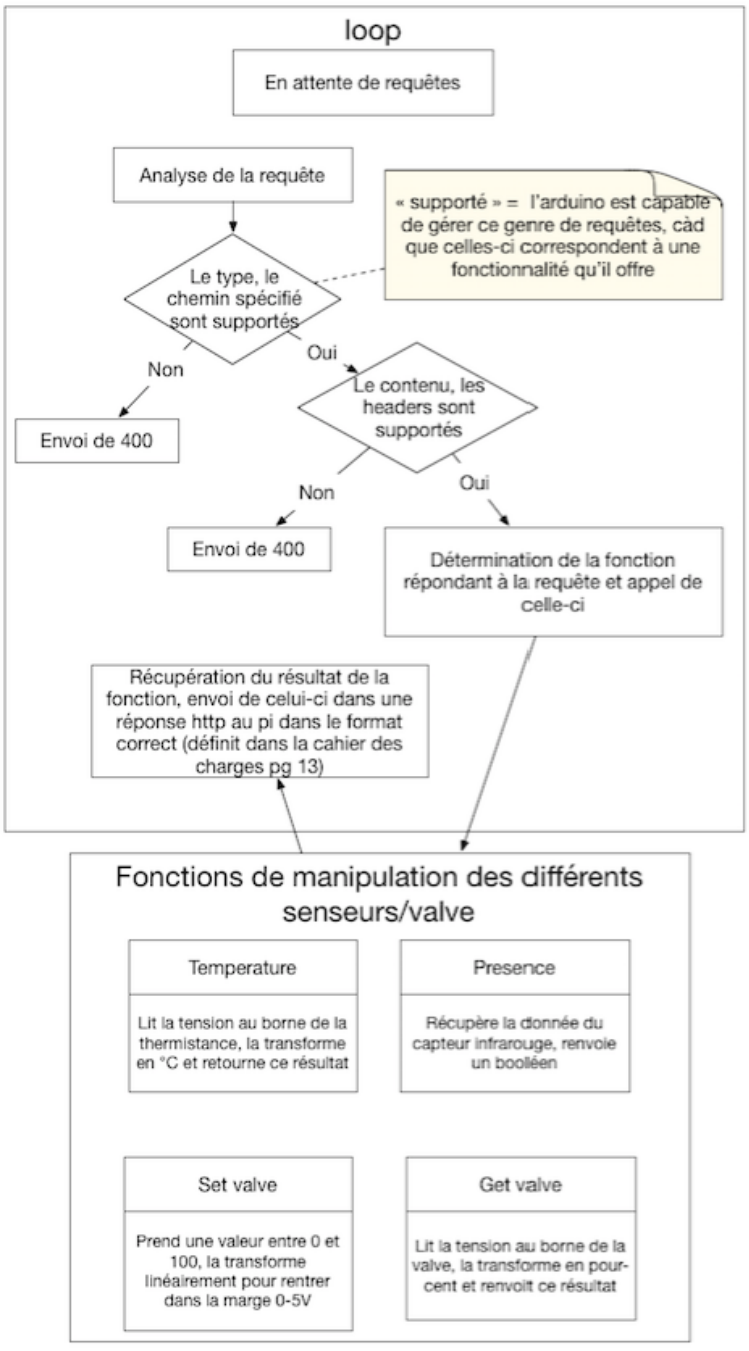


FIGURE 3.3 – Schéma du comportement d'un arduino

4 Thermostat central

1 Serveur

Cette partie est dédiée à la description du module `ThermoServer` dont le but principal est de gérer les Arduino connectés (dès la connexion et pendant la récolte des mesures). Il a été implémenté de façon à répondre strictement à la description du protocole de communication du cahier des charges. Son fonctionnement global est synthétisé sur la Figure 4.1. L'implémentation est telle qu'il est très facile de créer et de gérer de nouveaux types de senseur (autres que ceux de type « thin » et « outside » demandés dans le cahier des charges).

1.1 Implémentation

Le protocole de communication s'appuie sur un ensemble de classes qui interagissent entre elles, chacune effectuant une tâche bien précise (décrites brièvement dans le diagramme de classes présenté à la Figure 4.2 [1]). Chacune d'elles sont détaillées dans les sections suivantes.

1.1.1 `ThermoServer`

`ThermoServer` est la classe principale de ce serveur. Son but est de s'occuper simultanément de l'enregistrement des Arduinos, de la récupération et de l'enregistrement de leurs mesures dans une base de données et de la programmation de l'horaire de contrôle des vannes thermostatiques connectées. Ces fonctions sont également réparties sur les autres classes de ce module ainsi que sur les modules `Data`, `Utils` ainsi que tous ces gerant l'aspect réactif et l'intelligence du thermostat.

Plus techniquement, cette classe est initialisée au démarrage du Raspberry et lance un premier thread consacré au serveur enregistrant les Arduino qui tentent de se connecter (à l'aide de la classe `RequestRouter` décrite dans la partie 4), et un second récoltant périodiquement les mesures renvoyées par les thermostats auxiliaires connectés et les stockant dans une base de données dédiée. Cette classe envoie également les requêtes de contrôle des vannes thermostatiques.

1.1.2 `RemoteDevice`

Cette classe sert d'interface de communication pour les Arduino. Une instance de celle-ci est créée à chaque connexion d'un nouveau Remote Device et permettra d'envoyer chaque requête pour récupérer chacune de ses mesures et pour contrôler ce qui est modulable (les vannes par exemple).

De façon plus technique, cette classe est décrite par une adresse IP et un port permettant la communication avec l'Arduino, par un nom identifiant ces mesures et une liste de senseurs qui sont branchés sur ce thermostat auxiliaires. Cette liste permet de créer tout type de Remote Device, composé de plusieurs senseurs, tels qu'une thermistance, un capteur de présence ou même

1. Comme mentionné précédemment, les diagrammes de classes présentés dans ce rapport s'inspirent des règles proposés par l'UML mais ne sont pas strictement corrects du point de vue de ce langage. Leur but étant principalement documentaire, une trop grande rigueur n'a pas été jugée utile.

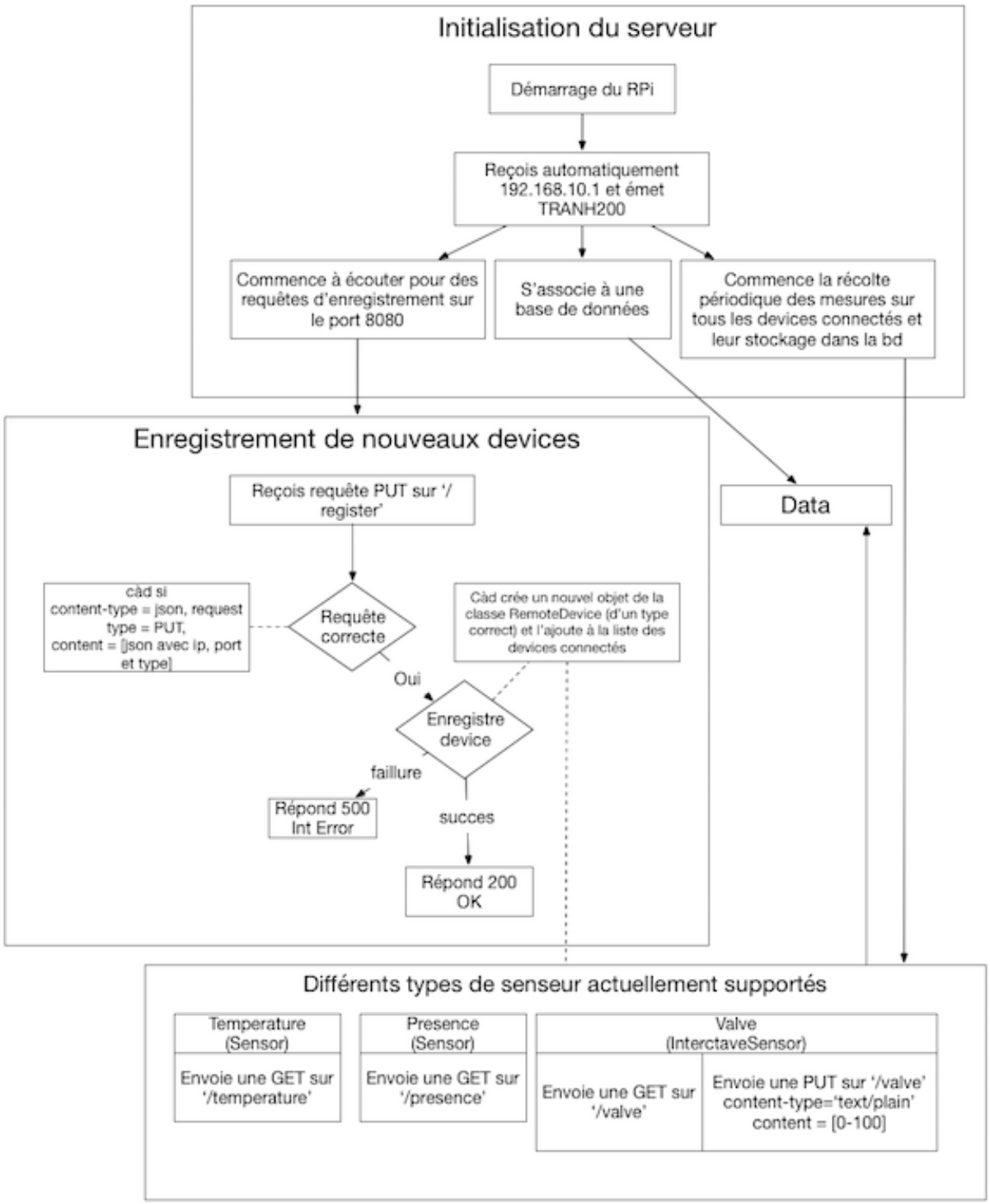


FIGURE 4.1 – Schéma du comportement du serveur

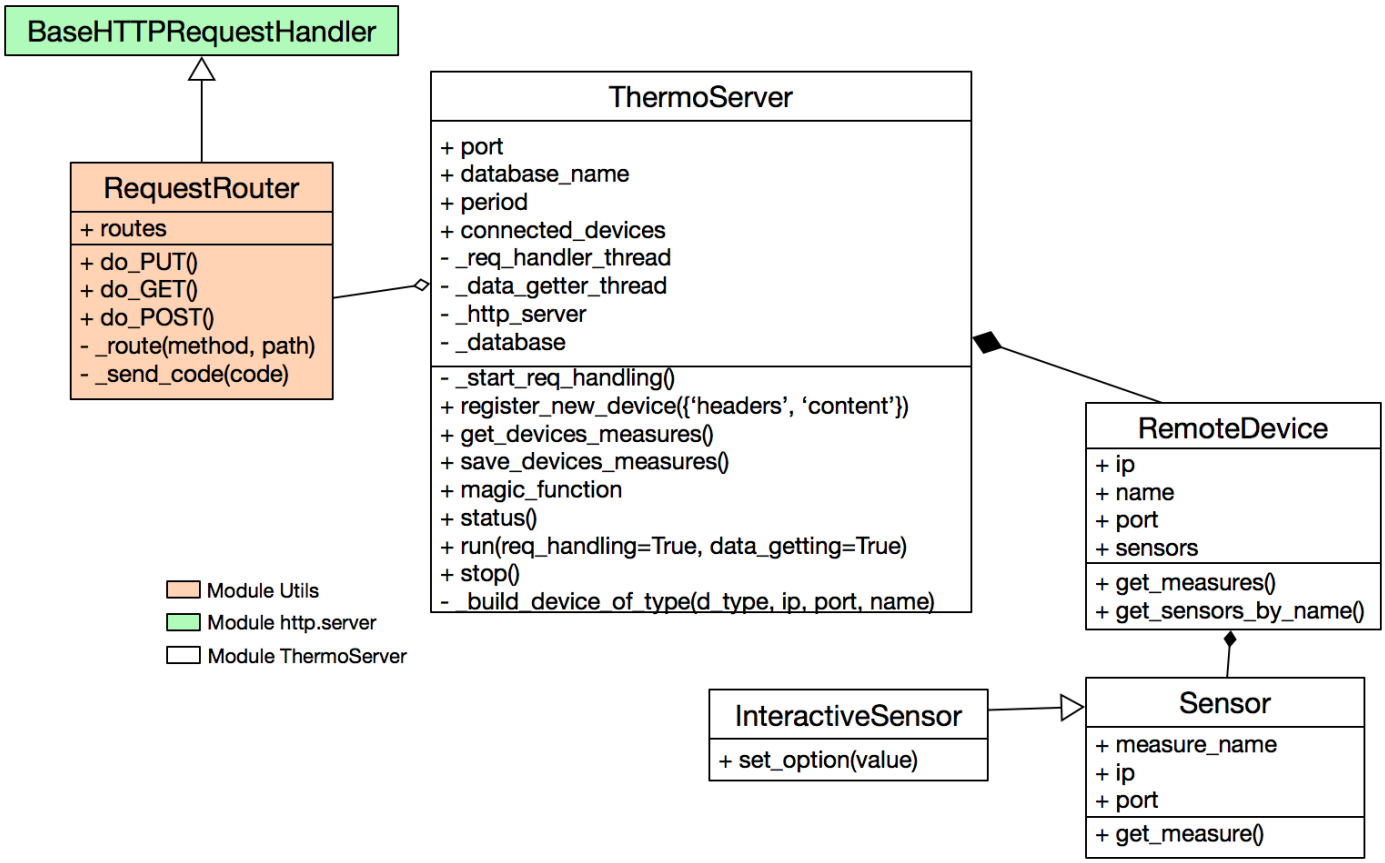


FIGURE 4.2 – Diagramme de classes du module ThermoServer

d'humidité, ou tout autre type de mesures imaginables et compatibles avec le circuit et le code de l'Arduino.

Grâce à cette implémentation modulaire des Remote Devices, il est possible de créer des objets connectés de tout type et pas seulement de type "thin" ou "outside" comme décrit dans le cahier des charges. L'évolution et l'expansion du projet est rendue beaucoup plus facile sans pour autant compliquer l'implémentation de ce qui est demandé.

1.1.3 `Sensor`

Une instance de cette classe permet simplement de récupérer la mesure du senseur correspondant. Par exemple, si l'on imagine un Arduino sur lequel est connecté une thermistance et configuré pour renvoyer la température ambiante en réponse à une requête de type `GET` sur le chemin `temperature`, cet objet de type `Sensor` permettra d'envoyer cette requête et de récupérer la température.

Dans le cadre du protocole de communication implémenté ici, un objet de type `Sensor` est créé pour chaque senseur physique branché et configuré à chaque Remote Device connecté au serveur.

1.1.4 `InteractiveSensor`

Cette classe hérite de la classe `Sensor` décrite ci-dessus en ajoutant la possibilité d'envoyer une requête de type `PUT` à un senseur pour le contrôler à distance. Typiquement, cette classe

permet de récupérer ET de modifier l'état d'ouverture d'une vanne thermostatique. Dans le cadre du projet, elle est typiquement associée à la valve.

1.1.5 `LocalSensor`

Cette classe possède le même comportement qu'une `InteractiveSensor` dans le sens ou une de ses instance elle représente une mesure utilisée par un Remote Device. La différence ici c'est que cette mesure est stockée sur le serveur et prend la forme d'une variable.

2 Gestion des données

2.1 Introduction

Afin d'implémenter une analyse intelligente des habitudes de l'utilisateur, un système de stockage de données est nécessaire. Deux options sont envisageables. Soit une structure sous forme de fichier, soit une autre utilisant les bases de données. Des recherches ont été menées pour déterminer quel système serait le plus adéquat. Il en est ressorti que les bases de données correspondent mieux à la problématique. En effet, celles-ci sont plus rapides, plus simples à manipuler et s'adaptent facilement à différentes tailles de jeux de données. De plus, en Python, de nombreuses bibliothèques les concernant existent, et certains types de bases de données, comme par exemple SQLite, y sont intégrés par défaut. Les raisons citées précédemment ont dès lors mené à une décision en faveur des bases de données.

2.2 Choix du gestionnaire base de données

Il existe différents SGBD [2], chacun ayant ses avantages et inconvénients. Une analyse comparative des plus populaires a été réalisée et SQLite a été choisi. La Table 4.1 justifie cette décision.

Caractéristiques	SQLite	MySQL	PostgreSQL
Installation	Module SQLite déjà implémenté dans Python.[31]	Installation d'un serveur MySQL nécessaire.[21]	Installation de PostgreSQL nécessaire.[30]
Types de données supportés	NULL, INTEGER, REAL, TEXT, BLOB.[33]	Complet, accepte également des entrées de type « Date ».[20]	Complet, accepte également des entrées de type « Date ».[26]
Stockage de la base de donnée	La base de donnée est condensée en un seul « fichier » stocké directement dans la mémoire (Du Raspberry dans ce cas).[34]	La base de donnée est stockée sur un serveur au quel doivent se connecter les utilisateurs.[19]	La base de donnée est stockée sur un serveur au quel doivent se connecter les utilisateurs.[27]

Table 4.1 – Tableau comparatif des différents SGBD étudiées

2.3 Implémentation

L'implémentation du module `Data` consiste en trois classes, `ThermoDB`, `QueryHandler` et `ThermoQueryHandler` ayant chacune un rôle distinct. Elle est construite selon une structure de

2. Système de Gestion de Base de Données

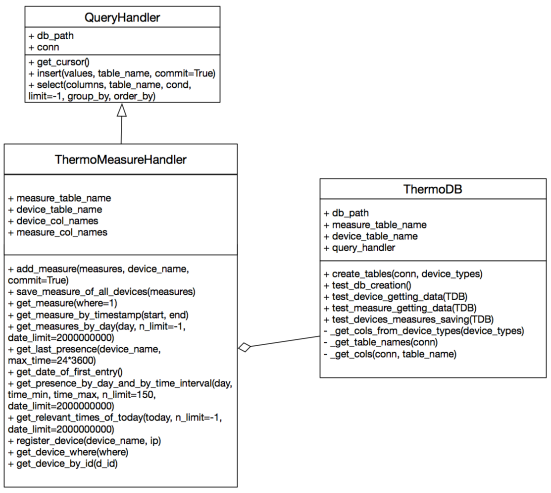


FIGURE 4.3 – Diagramme de classe du module `Data`

base de données formée de deux tables *measure* et *device* jointes par une relation many-to-one (voir Figure 4.3)

2.3.1 `ThermoDB`

Cette classe s'occupe de la création (processus lancé à l'initialisation du serveur) ou de la récupération (et de la vérification de la validité) d'une base de donnée existante. Ce qui peut être utile en cas de redémarrage du serveur.

2.3.2 `QueryHandler`

Il s'agit d'une classe pseudo-abstraite [3] définissant les fonctions de récupération et d'enregistrement de données génériques. Les définir permet de centraliser l'accès à la base de données et les éventuelles erreurs qui pourraient en découler. Cela permet de faciliter le débogage et les tests.

2.3.3 `ThermoQueryHandler`

Cette classe hérite de la précédente et définit des méthodes spécifiques à la manipulation des données relatives au thermostat.

3 Régulation en boucle fermée

3.1 Le choix du régulateur

Un des objectifs du régulateur est d'atteindre une valeur consigne, qui est dans ce cas-ci une température désirée. Un autre de ses objectifs est d'atteindre cette objectif avec stabilité (Sans oscillations autour de cette valeur) et ceci le plus rapidement possible. Il n'est en effet pas intéressant d'avoir un régulateur qui permette à la température de constamment varier [4] ou qui

3. Dans le sens où elle a été conçue pour être héritée

4. Ce qui consomme de l'énergie d'éteindre et rallumer le système de chauffage. Dans le cas d'un thermostat, on parle du phénomène de pompage

n'atteint pas la valeur consigne d'une manière optimale. 3 types de régulateurs ont été comparés : le régulateur "On/Off", le régulateur "PID" [5] et le régulateur "LQR" [6] [7] (Voir Tableau 4.2).

Caractéristiques	Régulateur On/Off	Régulateur PID	Régulateur LQR
Performance	-	++	+++
Stabilité	-	++	++
Facilité d'implémentation	+++	++	-

TABLE 4.2 – Tableau comparatif des différents régulateurs

Il clairement que le régulateur PID est le plus adéquat. En effet, il a été étudié que cette méthode est également quantitativement plus intéressante [8] [9].

3.2 Régulateur PID

3.2.1 Relation mathématique

Le régulateur PID peut être résumé par la relation :

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t) \tag{4.1}$$

où t représente le temps, e(t) représente l'erreur [10] et u(t) sera la sortie de notre régulateur qui dépendra des 3 constantes K_p,K_i et K_d, respectivement constantes de la proportionnelle, de l'intégrale et de la dérivée. Celles-ci doivent être déterminées et ce protocole sera explicité dans les sections qui suivent.

3.2.2 Ajustement des constantes

Deux voies sont possibles pour l'ajustement des constantes. La première consiste en l'utilisation de calculs d'équations de chaleur, la deuxième permettant plutôt de les obtenir expérimentalement. Elles s'obtiennent en les déterminant une à une successivement. En effet, il faut commencer par utiliser un régulateur P proportionnel et de régler la constante du gain proportionnel en fonction du résultat attendu. Une fois fixé, on utilise un régulateur PI pour fixer la constante du gain intégral et enfin un régulateur PID pour fixer la troisième constante. L'objectif final étant d'obtenir une courbe comme celle de la figure 6.1, sans aucune oscillation.

Il est cependant beaucoup plus judicieux que ces constantes se déterminent automatiquement et qu'elles s'adaptent de manière pro-active. De plus, il faut que ce système soit le plus *user-friendly* possible afin que tout le monde puisse profiter du thermostat sans aucune connaissance

5. PID pour "Proportional Integral Derivative"

6. LQR pour "Linear-Quadratic Regulator"

7. Bien qu'il existe d'autres régulateurs, ce dernier n'étany pas le sujet central de ce travail il ne semble pas intéressant d'en comparer un plus grand nombre. A ce titre, peuvent être cités comme régulateur : "Fuzzy control", "Process reaction method", "Fuzzy PID control", "Adaptive control" ou encore "Sliding Mode Control"

8. Facilité d'implémentation : Notamment au niveau de la paramétrisation du jeu de constantes utilisées

9. A nouveau, l'analyse de graphique de performance de performance n'étant pas le sujet principal de ce travail, le lecteur est renvoyé vers la bibliographie pour plus de détails sur ces études

10. Ici, l'erreur représente la différence entre la température demandée (valeur consigne) et la température mesurée

en physique ou informatique.

Un algorithme d'initialisation automatique et optimisé des constantes a été développé. Ce dernier permet comme explicité précédemment d'obtenir, après plusieurs jours d'utilisations, une approche assez fidèle des constantes « réelles ». Les explications associées sont présentes dans la section qui traite de la partie 4.2.

3.2.3 Implémentation du PID dans le serveur central

Deux classes principales sont implémentées : `PID` et `PIDHandler`.

La première possédant les méthodes nécessaires au calcul de l'erreur (Température effective - Température cible), de la dérivée de l'erreur et de l'intégrale de l'erreur. Ces trois valeurs sont les valeurs présentes dans l'équation 4.1. Deux autres méthodes sont également présentes : `update` et `set_constants`. La première permet d'actualiser les attributs de la classe PID concernant la température (visée/effective), la date ou l'ouverture de la valve tandis que la deuxième s'occupe exclusivement de gérer les constantes du régulateur : K_p, K_i, K_d.

Le rôle de la deuxième sera détaillé à la section 4.2 du chapitre 5.

4 Module `Utils`

Ce module contient trois classes principales : `RepeatingTimer`, `RequestRouter` et `TimeOperator`.

Il a pour but de contenir des classes ou plus généralement des outils qui sont complètement indépendants du projet (mais utiles à celui-ci) et qui pourraient donc être facilement utilisables pour un quelconque autre projet. Outre le fait que la séparation des classes génériques leur permet d'être réutilisées dans d'autre projet, se pliant ainsi aux règles de bonnes pratiques, la construction d'un module séparé contenant ces classes permet d'alléger les autres modules et de les rendre plus compréhensible.

Ces deux classes sont donc construites de façon générique et accomplissent des taches élémentaires.

4.1 `RepeatingTimer`

La seule fonction de cette classe (héritant de la classe `Timer` du module `threading` de python) est d'appeler une fonction spécifiée à intervalle régulier. Elle est utile, dans ce projet, à la collecte régulière des mesures du serveur sur les remote devices connectés.

4.2 `RequestRouter`

Cette classe, héritant de `BaseHTTPRequestHandler` fournie dans le module `http.server` de python 3, a pour but d'écouter sur un port spécifié pour d'éventuelles requêtes et de les "router" vers un callback spécifié en fonction de la méthode http et du chemin de la requête.

Elle est totalement générique puisque n'importe quelle requête peut déclencher n'importe quelle fonction (pour peu qu'elle soit configurée correctement) et peut-être utilisée dans n'im-

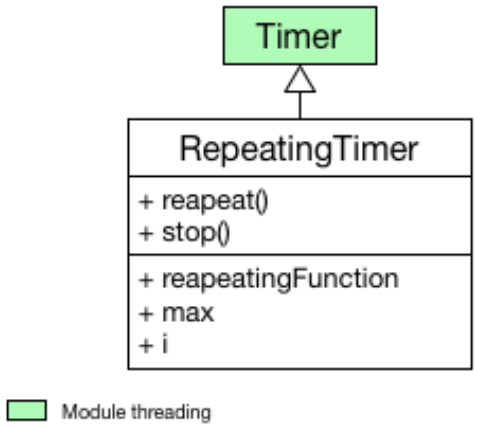


FIGURE 4.4 – Diagramme de classe pour `RepeatingTimer`

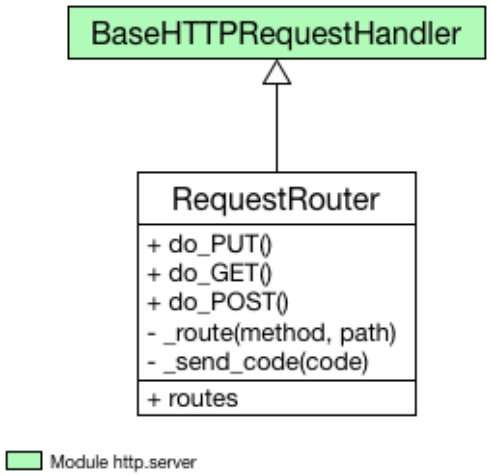


FIGURE 4.5 – Diagramme de classe pour `RequestRouter`

porte quel cas nécessitant un serveur.

Dans le cadre spécifique de ce projet, cette classe est utile à la réception et au traitement des requêtes d'enregistrement des remote devices.

4.3 `TimeOperator`

Cette classe possèdes des méthodes statiques[11] ayant pour but d'effectuer les opérations temporelles nécessaires aux modules intelligents présentés à la section 5. Elle est détaillée à la figure 4.6.

5 Débogage

Un système de logging est implémenté dans tous les modules pour garder une trace de chaque éventuelle erreur, dysfonctionnement ou anomalie lors de l'exécution du serveur. Le dépannage est

11. Cette classe ne présentait pas d'intérêt à devoir être instanciée.

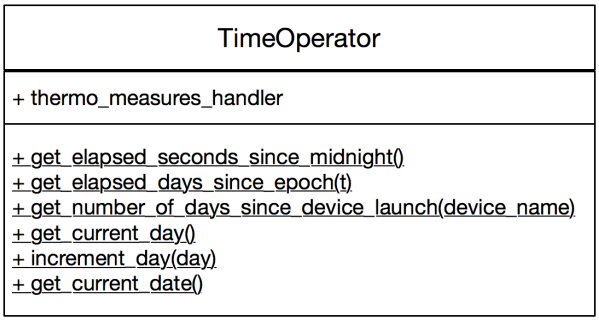


FIGURE 4.6 – Diagramme de classe pour `TimeOperator`

alors simplifié lors d'une utilisation réelle et le débogage est plus rapide lors du développement dès lors que l'origine des erreurs est plus facilement identifiable. Il est ainsi possible dès le lancement du serveur d'indiquer quels types (niveau de gravité) de messages doivent être affichés et/ou stockés dans un fichier journal.

5 Intelligence du code

1 Introduction

Cette partie concerne les modules utilisés par le serveur afin de rendre le thermostat prédictif. Elle présente trois objectifs principaux qui ont pour but de rendre le système autonome. Un premier module calcule des probabilités en se basant sur la base de données et fournit des prédictions de présence, un second optimise les constantes du contrôleur PID afin qu'il s'adapte à n'importe quelle pièce et finalement, un dernier y fait appel pour évaluer le temps nécessaire pour atteindre la température désirée.

2 Critères d'intelligence

L'objectif principal d'un thermostat intelligent est de minimiser les dépenses énergétiques liées à la régulation thermique tout en assurant à ses utilisateurs un confort au minimum égal à celui offert par un dispositif traditionnel.

Ces deux aspects sont étudiés ici. Tout d'abord il est nécessaire de définir les notions de consommation énergétique et de confort thermique [1]. La première a été standardisée [5] et dépend de nombreux paramètres (humidité, température, aération, ...). Cependant, dans le cadre de ce projet et par souci de simplification, seule la température intérieure est étudiée. Il a donc été supposé que le confort maximal est atteint lorsque la différence entre la température actuelle et la température désirée par l'utilisateur est minimale [2].

Concernant la consommation énergétique, il a été considéré qu'elle serait minimisée à condition de raccourcir un maximum les périodes de chauffe [3] ainsi qu'en réduisant l'oscillation de la température désirée autour de celle effective.

3 Prédiction de présence

3.1 Introduction

Pour que le système soit prédictif, il doit être capable d'estimer les chances de présence de l'utilisateur dans une pièce à un instant donné. Afin de rationaliser cette démarche, il est utile d'aborder l'analyse prédictive, celle ci usant entre autre de probabilités et de statistiques. Le code présenté dans cette section a pour but de générer et manipuler un modèle mathématique consistant en une fonction associée à une probabilité en tout temps. La méthode utilisée sera : « l'interpolation polynomiale » [6]. Cela consiste en la représentation de la fonction par un

1. Ces définitions ont été définies sur base d'articles traitant de ce sujet.
2. Cet objectif sera atteint au moyen de la régulation PID
3. Ceci au moyen des prédictions de présence.

polynôme dont les constantes sont calculées par analyse statistique des mesures récoltées et stockées dans la base de données. Ces dernières seront comparées au valeurs effectivement observées.

D'autres méthodes comme les séries chronologiques ou même les séries de Fourrier auraient pu être utilisées, mais l'utilisation d'un polynôme semblait être plus simple et suffisant. L'implémentation du calcul automatique des coefficients de Fourrier et leur actualisation aurait pu se révélée bien plus complexe et fastidieux.

3.2 Modélisation mathématique

Le but de ce calcul est d'obtenir la fonction de probabilité introduite précédemment. Sa structure sera polynomiale afin de calculer ses constantes par interpolation.

$$p : \mathbb{R}^+ \to [0;1] : t \mapsto At^5 + Bt^4 + Ct^3 + Dt^2 + Et + F \tag{5.1}$$

Comme présenté dans l'équation 5.1, c'est une fonction de degré 5. Ce choix arbitraire a pour but de concilier une interpolation [4] de qualité suffisante avec un temps de calcul acceptable [5].

La méthode appliquée pour calculer les constantes optimales consiste en une minimisation du carré [6] de la somme des différences entre le polynôme et la probabilité à approcher en tout temps. Si la probabilité mesurée est exprimée comme :

$$E(t_i) = \frac{n(t_i)}{N(t_i)} \tag{5.2}$$

Avec $n(t_i)$ correspondant au nombre de présence et $N(t_i)$ au nombre total de prises de mesures à l'instant t_i [7]. Cette grandeur est toujours comprise entre 0 et 1, ceci justifiant l'utilisation de l'aspect probabiliste du problème.

Dès lors, l'expression à minimiser prend cette forme :

$$\sum_{i=0}^{n} (p(t_i) - E(t_i))^2 \tag{5.3}$$

En égalant le gradient de cette expression au vecteur nul, et en posant $k_j = \sum_{i=0}^{n} t_i^j$ et $l_s = \sum_{i=0}^{n} E(t_i) t_i^s$, le système à résoudre est obtenu :

$$\begin{cases} Ak_{10} + Bk_9 + Ck_8 + Dk_7 + Ek_6 + Fk_5 = l_5 \\ Ak_9 + Bk_8 + Ck_7 + Dk_6 + Ek_5 + Fk_4 = l_4 \\ Ak_8 + Bk_7 + Ck_6 + Dk_5 + Ek_4 + Fk_3 = l_3 \\ Ak_7 + Bk_6 + Ck_5 + Dk_4 + Ek_3 + Fk_2 = l_2 \\ Ak_6 + Bk_5 + Ck_4 + Dk_3 + Ek_2 + Fk_1 = l_1 \\ Ak_5 + Bk_4 + Ck_3 + Dk_2 + Ek_1 + Fk_0 = l_0 \end{cases} \tag{5.4}$$

Les constantes calculées permettent d'obtenir la fonction qui statistiquement est la plus fidèle à la réalité expérimentale.

4. Plus le degré du polynôme est élevé, meilleure sera l'interpolation.
5. Si le polynôme est de degré n, la matrice du système d'équation à résoudre sera d'ordre n+1.
6. Afin de ne pas avoir d'annulation de valeurs
7. t_i sera considéré par la suite comme un intervalle, ce qui justifie cette modélisation.

3.3 Implémentation

La gestion de la probabilité de présence se décompose en deux classes. Une première, `ProbabilityModel`, gère la fonction mathématique de probabilité de manière basique et est capable de calculer la probabilité à un instant donné. Elle peut également mettre à jour les constantes de la formule après que le calcul aie été effectué dans la seconde. Cette dernière s'appelle `ProbabilityModelHandler` et a pour but de calculer la solution du système d'équations 5.4. Les liens entre ces classes et leur fonctionnement sont explicités sur le diagramme de classes 5.1.

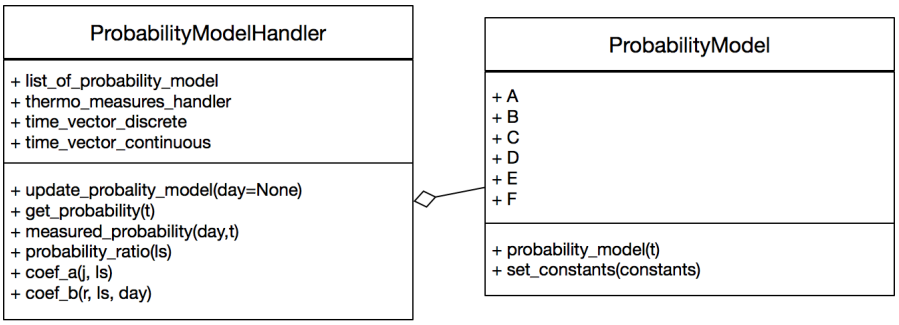


FIGURE 5.1 – Diagrammes des classes `ProbabilityModel` et `ProbabilityModelHandler`

3.3.1 Récupération des données et gestion temporelle

Après une réflexion concernant la distribution temporelle des modèles, il est apparu que le meilleur compromis entre confort d'utilisation et simplicité d'implémentation est de générer un modèle par jour de la semaine. En effet, il est plausible d'estimer que le rythme de vie de l'utilisateur, et donc sa fréquence de présence, est répétitif selon le jour de la semaine. Toutefois, en envisageant que ça ne soit pas le cas, le modèle resterait fonctionnel, il convergerait juste moins rapidement vers la courbe optimale.

Afin de regrouper suffisamment de données de présence et de leur donner un sens d'un point de vue statistique, les 24 heures d'une journée ont été discrétisées en 96 éléments [8] d'un quart d'heure. En effet, sans cette opération, il serait stochastiquement impossible que deux mesures soient prises le même jour de la semaine précisément à la même heure [9]. De ce fait, une méthode récupérant les données encodées un jour de la semaine et appartenant au quart d'heure considéré a été codée dans le module `Data`. La suite de l'implémentation repose sur les données qu'elle récupère.

3.3.2 Résolution

`ProbabilityModelHandler` possède deux méthodes principales : *update_probality_model*, qui permet de mettre à jour les coefficients du polynôme de probabilité afin de constamment pouvoir approcher la réalité un maximum. Cette méthode a donc pour objectif de résoudre le système 5.4 afin d'en déduire les constantes $A\ B\ C\ D\ E\ F$. La classe contient également *get_probabiblity* afin de récupérer la probabilité qu'il y ait quelqu'un à un instant donné. Cette méthode à pour but de « lire le graphique » dessiné par la fonction de probabilité de présence. Cette dernière valeur sera ensuite comparée avec un seuil pour déterminer si la pièce doit être chaude pour cette heure là où non.

8. Ces éléments représentent les t_i présentés dans la section 3.2.

9. L'heure est encodée au format EPOCH dans la base de données et comporte donc 6 chiffres après la virgule.

4 Prédiction du temps de chauffe

4.1 Introduction

Il est essentiel de pouvoir déterminer le temps exact nécessaire pour chauffer une pièce afin de conserver un confort maximum d'utilisation. Comme mentionné dans la section précédente, l'objectif est d'avoir une pièce chauffée lorsque la valeur de la probabilité de présence passe une valeur seuil. Il apparaît vite que « simplement » enclencher le chauffage lorsque la fonction passe ce seuil n'est pas la manière optimale de procéder.

Une prédiction du temps de chauffe propre à chaque pièce doit donc être réalisée. Cette opération systématique est donc réalisée à partir de l'algorithme suivant. Il aura pour objectif de déterminer le temps t nécessaire pour chauffer une pièce d'une température T_0 à une température T_f.

Cette manière de procéder permet deux grands avantages :

— Ne pas devoir utiliser d'équations de chaleur prenant en compte les pertes thermiques de la pièce

— Ne demande à l'utilisateur de n'effectuer aucune manipulation

L'algorithme de prédiction de présence travaillera donc en décalage, si le temps de chauffe est : t_c, alors le programme analysera la possibilité d'une présence à $t + t_c$ afin que le chauffage aie le temps de chauffer la pièce.

4.2 Calcul et optimisation des constantes du contrôleur PID

Le dispositif pouvant être théoriquement installé dans n'importe quelle pièce et actionner des chauffages de puissances différentes, il doit être capable de calculer l'ouverture optimale de la vanne, et donc par extension les constantes du régulateur PID.

4.2.1 Modélisation mathématique

Du point de vue du PID, un accroissement de température de t_0 à t_k s'exprime :

$$\Delta T = T(t_k) - T(t_0) = \int_{t_0}^{t_k} PID(t)dt \Leftrightarrow T(t_k) = T(t_0) + \int_{t_0}^{t_k} PID(t)dt \tag{5.5}$$

En effet, il n'est pas nécessaire de tenir compte des caractéristiques thermodynamiques de la pièce car le PID ne doit pas modéliser de comportement thermique. Les effets de ces paramètres sont pris en compte dans ses trois constantes.
Si on considère deux instants proches t_i et t_{i-1}, il est acceptable de considérer que l'équation 5.5 peut être approchée par :

$$T(t_i) \approx T(t_{i-1}) + PID(t_{i-1}) * (t_i - t_{i-1}) \tag{5.6}$$

Le but poursuivi par le régulateur PID est d'atteindre une température cible T_T le plus rapidement possible et ensuite de la maintenir. En d'autres mots, il minimise en permanence la différence entre la température actuelle et la température visée. L'expression à minimiser vaut :

$$\sum_{i=1}^{n} (T(t_{i-1}) + PID(t_{i-1}) * (t_i - t_{i-1}) - T_T(t_i))^2 \tag{5.7}$$

Ce qui donne, en injectant l'expression d'un PID :

$$\sum_{i=1}^{n}(T(t_{i-1}) + K_p e(t_{i-1}) + K_i \int_{t_0}^{t_{i-1}} e(t)dt + K_d \frac{d}{dt} e(t_{i-1}) * (t_i - t_{i-1}) - T_T(t_i))^2 \tag{5.8}$$

En dérivant trois fois l'équation par rapport aux constantes K_p, K_i et K_d, de la même manière que pour la prédiction de présence, on obtient le système suivant :

$$\begin{cases} \sum_{i=1}^{n} e(t_{i-1}) * (T(t_{i-1}) + K_p e(t_{i-1}) + K_i \int_{t_0}^{t_{i-1}} e(t)dt + K_d \frac{d}{dt} e(t_{i-1}) * (t_i - t_{i-1}) - T_T(t_i)) = 0 \\ \sum_{i=1}^{n} \int_{t_0}^{t_{i-1}} e(t)dt * (T(t_{i-1}) + K_p e(t_{i-1}) + K_i \int_{t_0}^{t_{i-1}} e(t)dt + K_d \frac{d}{dt} e(t_{i-1}) * (t_i - t_{i-1}) - T_T(t_i)) = 0 \\ \sum_{i=1}^{n} \frac{d}{dt} e(t_{i-1}) * (t_i - t_{i-1}) * (T(t_{i-1}) + K_p e(t_{i-1}) + K_i \int_{t_0}^{t_{i-1}} e(t)dt + K_d \frac{d}{dt} e(t_{i-1}) * (t_i - t_{i-1}) - T_T(t_i)) = 0 \end{cases} \tag{5.9}$$

A nouveau, comme pour la modélisation de la probabilité de présence, les valeurs des constantes qui minimisent cette différence sont le résultat du système 5.9.

4.2.2 Implémentation

La classe `PIDHandler` présentée à la section 3.2.3 est utilisée pour résoudre ce système. Une méthode principale `update_pid_model` fait appel à de nombreuses fonctions qui sont utilisées afin de générer les coefficients des matrices à résoudre, et des algorithmes d'analyse numérique ont été implémentés [10][22].
Ce calcul fait appel à une fonction agissant sur la base de données afin de récupérer toutes les entrées reçues le jour où une optimisation est lancée [11] C'est dans cette optique qu'une colonne supplémentaire, la « target_temp », y a été ajoutée.

4.3 Calcul des constantes thermodynamiques

Le but de cette section est d'étudier le comportement thermique d'une pièce. Pour ce faire, le code doit analyser ses caractéristiques thermodynamiques.

4.3.1 Modélisation mathématique

En définissant P_f et P_d, respectivement les puissances fournies et dissipées dans la pièce considérée, il est possible de modéliser l'évolution de la température comme suit :

$$T(t) = T(t_0) + \int_{t_0}^{t} P_f(\tau)d\tau - \int_{t_0}^{t} P_d(\tau)d\tau \tag{5.10}$$

P_f peut être exprimée comme un coefficient [12] modulé par la valeur du PID. P_d quant à elle, s'apparente à la loi de Fourier pour les pertes de chaleur aux travers de parois[13]. Elle s'exprime

10. Afin de mettre en place une résolution propre du calcul intégral et différentiel.
11. Cette classe a été pensée de manière à exécuter ses méthodes une fois par jour.
12. Correspondant à la puissance maximale du chauffage.

donc comme une constante thermique [13] multipliée par la différence de température entre la pièce et l'extérieur. En tenant compte de ces relations, on obtient :

$$T(t) = T(t_0) + P_{max} \int_{t_0}^{t} PID(\tau)d\tau - \sigma \int_{t_0}^{t} (T(\tau) - T_{ext}(\tau)d\tau \tag{5.11}$$

Le but de cette modélisation est de déterminer ces deux constantes. Pour ce faire, il faut répéter la prise de mesures deux fois. Dès lors un système de deux équations est obtenu.

4.3.2 Implémentation

Ce module n'a été implémenté que basiquement, et ne présente pas de structure claire à présenter. Toutefois, cette classe accède à la base de données afin de récolter les données brutes nécessaires au calcul des coefficients du système. Tout comme les autres codes intelligents, il a été pensé de telle sorte à ce s'exécuter régulièrement afin de s'adapter si des changements d'environnement ont lieu.

4.4 Conclusion

Ces nombreux calculs ont pour but de permettre au serveur de prévoir le temps que va mettre une pièce pour atteindre une température, en connaissant les températures actuelle et extérieure. Pour ce faire, il interroge le `PID`, qui en permanence évalue cette valeur. Ce dernier est en mesure de le faire car il connaît le modèle thermique de la pièce dans laquelle le Remote Device auquel il est lié est situé. Il simule un ordre de chauffe, et renvoie le temps Δt qu'il faut pour atteindre la température désirée.

Ce module n'est pas encore fonctionnel, et n'a donc pas été testé entièrement. Le calcul des constantes du PID ne donne pas de résultats corrects. Il a été envisagé d'appliquer une méthode tout à fait différente, celle de Ziegler-Nichols[1]. Elle n'a toutefois pas été retenue dans un premier temps car son implémentation semblait trop complexe. Cependant, elle semble à présent être celle à utiliser si il fallait rendre le code fonctionnel.

5 Fonctionnement du code intelligent

Des éléments intelligents sont présents dans deux Threads. Cette section va présenter comment le serveur va les utiliser.

5.1 Thread principal

Dans la classe `ThermoServer`, une méthode principale, `magic_function`, tourne en permanence. Celle-ci, en plus de s'occuper de la récolte et du stockage des données pour chaque remote device, va évaluer la nécessité de chauffer la pièce dans laquelle il se situe. Pour se faire, elle utilise le modèle de probabilité du jour ainsi que le temps de chauffe fourni par les simulations du `PID`.

Lorsque le serveur évalue la probabilité de présence en un instant, il la compare à une valeur seuil, la « Trigger value » [14]. Si celle-ci est inférieure à la probabilité actuelle, un ordre de chauffe est envoyé. Ceci pose un problème. En effet, cette manière de faire est en retard, car l'activation des radiateurs a seulement lieu au moment où un utilisateur est susceptible d'arriver. C'est pour

13. Equivalente à la tendance qu'a l'habitation à dissiper sa chaleur.

14. Conventionnellement, cette valeur vaut 0,5.

celà qu'il faut considérer le temps de chauffe de la pièce.
Si on considère t, l'instant présent, et Δt, le temps de nécessaire pour atteindre la température visée, la probabilité à considérer sera celle-ci :

$$p(t + \Delta t) \tag{5.12}$$

5.2 Thread d'optimisation

En parallèle, un thread de calcul et d'optimisation de toutes les constantes évoquées dans ce chapitre sera exécuté une fois par jour. De cette manière, le code s'adaptera si des changements interviennent dans les habitudes des utilisateurs ou dans les caractéristiques thermodynamiques d'une pièce [15].

15. Changement d'emploi, horaires différents, nouvelle isolation, installation de double-vitrage, nouveaux chauffages, déménagement,...

6 Stratégie de tests et validation

Une fois l'implémentation des codes finalisée et malgré leur fonctionnement apparent, il est indispensable de réaliser deux tâches.

La première tâche consiste à mettre sur pied une stratégie de tests et d'essayer toute une série de scénario pour voir comment réagissent le thermostat individuel et le thermostat central à ces différents stress tests. Est ce que la gestion des exceptions est opérationnelle ? Que se passe-t-il si le serveur central s'éteint brutalement ? Et qu'en est il des données envoyées si le matériel est défaillant ? Autant de questions auxquelles les tests doivent pouvoir répondre.

D'un autre côté, il faut également que le thermostat puisse être évalué par des critères objectifs. Il faut donc pour cela développer des indicateurs de qualité qui permettent de disposer d'arguments forts sur les avantages ou les désavantages du système développé.

Enfin, il est à noter que suite à des événements indépendants de notre volonté (Le module wifi a été défaillant pendant plusieurs semaines malgré son remplacement.), il n'a pas été possible de tester et observer des résultats pour tout ce qui devait être testé. Néanmoins, il a été décidé de quand même exposer la réflexion du groupe sur qui a voulu être implémenté comme tests.

1 Stratégie de tests

Chaque test sera explicité selon deux aspects, ce qui fut testé et les résultats qui furent attendus et/ou observés.

— Les composants du thermosat individuel : la prise de température par la thermistance, le détecteur de présence, l'ouverture des vannes de l'environnement physique, l'envoi ou la réception de requêtes au serveur central par le module WiFi. Tous ces tests ont été concluants. Notons que pour la thermistance, les résultats obtenus ont été comparé à la température obtenue par l'environnement physique. *(Notons tout de même une légère exception suite à la défectuosité du module Wi-fi, mais il s'avère que la source d'erreur se trouve être le module lui même et non le code)*

— Le serveur central : la connexion à la base de donnée, l'envoi de requêtes, la réception de requêtes d'enregistrement du thermostat individuel et de l'environnement furent testés. La création d'une base de donnée si elle est inexistante et l'utilisation d'une base de donnée si elle existe également. Les tests ont tous été concluants.

— Le régulateur de température (PID) sans initialisation automatique des constantes : l'objectif était de tester si le régulateur PID fonctionnait parfaitement si on déterminait expérimentalement les constantes du PID. Puisque le module WiFi ne fonctionnait déjà plus à ce moment là, il a été décidé de créer un script pour simuler un environnement (Voir annexe. Le résultat est que le PID permettait d'atteindre la valeur consigne et de stabiliser la température autour de cette valeur (voir Figure 6.1). Une remarque importante doit être signalée sur le simulateur. En effet, le gain en température a été codé comme étant proportionnel à l'ouverture de la vanne et la perte proportionnelle à la différence entre la température mesurée et la température extérieure. Il ne convient donc pas de tirer de

pertinence sur la lecture du temps passé en abscisses car elles dépendent de ces coefficients de proportionnalité. Notons enfin que la température initiale vaut 10 et la valeur consigne 20..

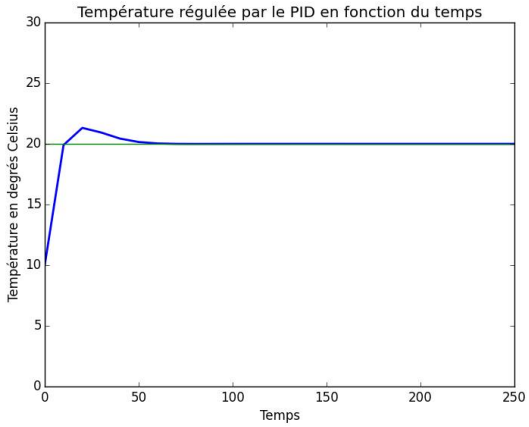


FIGURE 6.1 – Résultat du PID par simulation

— Le régulateur de température (PID) avec initialisation automatique des constantes : l'objectif était de tester si le régulateur PID était capable d'initialiser automatiquement ses constantes. Malheureusement, force est de constater que les résultats avec le simulateur n'étaient pas concluants. Un problème de modélisation théorique semble être à l'origine du problème.

— Modèle prédictif de présence : l'objectif était de voir si le serveur central était capable d'observer une courbe de probabilité à partir de données qui lui seraient envoyées. Là encore, le problème du module WiFi ne nous a pas permis de tester directement le code. Il a donc à nouveau été question d'utiliser un simulateur de données afin d'observer les résultats obtenus. Les résultats ont été concluants. La modélisation par interpolation polynomiale de degré 5 permet bien d'obtenir une courbe de probabilité (voir Figure 6.3 et 6.2). Il peut être utile de signaler que lorsque la probabilité de présence n'est pas comprise dans l'intervalle [0,1], elle est automatiquement ramenée à l'une de ces deux valeurs dans le code.

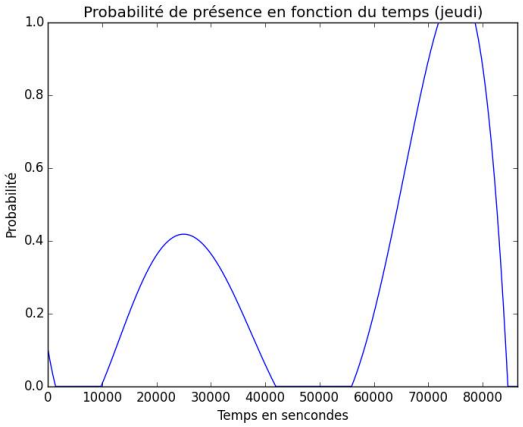


FIGURE 6.2 – Résultat du modèle de prédiction par simulation - Jeudi

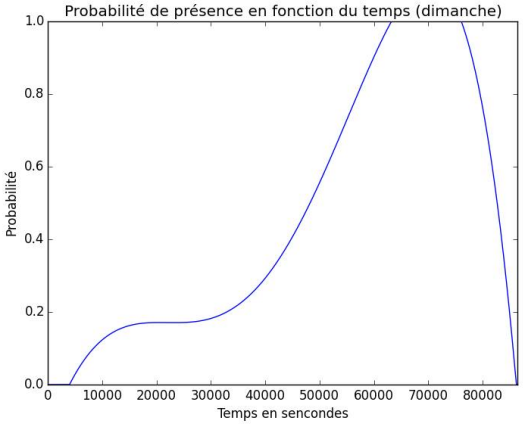


FIGURE 6.3 – Résultat du modèle de prédiction par simulation - Dimanche

2 Validation du modèle

Pour la validation du modèle, quatre questions seraient intéressantes à analyser en se positionnant en tant qu'utilisateur du thermostat intelligent. Il a en effet toutes les chances de se poser les questions suivantes :

1. L'intelligence du code permet-il de prédire efficacement la présence de l'utilisateur ?

2. Lorsque qu'une valeur consigne doit être atteinte, à quelle vitesse cette valeur est-elle atteinte ?

3. Une fois cette valeur atteinte, le régulateur est-il assez performant pour maintenir une température stable et ne pas osciller ?

4. Quel est le coût énergétique de ce régulateur en comparaison au "On/Off" algorithme ?

Étant donné les problèmes du module WiFi évoqué dans la section "Stratégie de test", il n'a été possible que d'évoquer les indicateurs tels qu'ils auraient été implémenté si cela avait été possible. Les solutions auraient respectivement été :

1. Le modèle est censé prédire une présence à chaque seconde pendant 24h. Il suffit de comptabiliser le nombre de fois où il a préchauffé une pièce alors qu'il ne fallait pas et le nombre de fois où il n'a pas préchauffé une pièce alors que quelqu'un arrivait et on divise la somme de ces résultats par le nombre d'observations totales faites sur une journée. Plus cet indicateur est bas et plus le modèle prédictif est bon.

2. Il faudrait lancer le programme et de récolter les données à intervalles réguliers afin de connaître le temps nécessaire au thermostat pour atteindre la valeur consigne. Il pourrait également être intéressant de connaitre la variation de ce temps de chauffe lorsque chaque constante du PID est modifiée.

3. Une méthode utile serait de calcul sommer tous les écarts de température et donc d'intégrer cette fonction au carré [1] sur tout l'intervalle où la température est sensée ne plus varier.

1. Afin d'éviter les annulations d'erreur

Un algorithme stable devrait avoir une valeur de l'intégrale proche de 0.

4. Une possibilité serait de considérer l'ouverture moyenne de la vanne au cours du temps pour les deux régulateurs. Cela en comptant que le fait d'allumer le chauffage consomme également plus d'énergie *(Pompage)*.

7 Fonctionnement de groupe

Afin d'atteindre les objectifs du projet le plus rapidement et le plus efficacement possible, il était important d'établir une bonne organisation du travail en groupe afin de maximiser la productivité du groupe et de répartir la quantité de tâches à effectuer par chacun de manière équitable. Quelques outils ont donc été mis en place dès la première semaine dans cette optique.

Pour commencer, le plus important était de définir comment les informations transiteraient. Un groupe privé sur Facebook a été choisi pour la communication informelle en dehors des réunions, l'avantage étant de pouvoir conserver un historique de ces conversations. Le transfert des documents, codes, parties de rapport,... s'effectue quant à lui via une Dropbox qui permet d'archiver et de partager tous ces fichiers de façon simple et efficace. Il a également été décidé que le rapport serait rédigé en LaTeX sur « Overleaf [1] », cette plate-forme permettant la modification en temps réel du document par tous les membres de l'équipe.

La fréquence des réunions fut fixée au rythme de deux par semaines. Une avec le tuteur du projet afin de le tenir au courant des avancées réalisées lors de l'autre réunion qui concerne uniquement les membres du groupe. Lors de ces rencontres hebdomadaires, chacun se doit de comprendre les avancée exposées par les autres membres du groupe afin d'obtenir des connaissances homogènes dans tous les domaines. Cette fréquence permet de ne pas oublier de travailler chaque semaine sur le projet. Il arrive que des réunions supplémentaires soient organisées en équipe lorsque le besoin s'en fait sentir ou qu'une deadline approche.

Ces réunions se tiennent, sauf cas de force majeure, à la Bibliothèque des Sciences et Techniques de l'ULB, principalement pour des raisons de proximité. Durant ces réunions, un membre est désigné et joue le rôle de secrétaire et d'animateur. La fusion de ces deux rôles a été décidée en raison de la taille du groupe. La tâche de de cette personne est d'envoyer l'ordre du jour aux autres membres au moins 24 heures avant la réunion, de l'animer ensuite et d'en rédiger le procès-verbal par après. Ce poste est attribué à une autre personne toutes les deux semaines sur base volontaire afin de ne surcharger personne de travail.

1. `https://www.overleaf.com/`

8 Perspectives d'évolution

Beaucoup d'hypothèse simplificatrice ont été faites jusqu'ici. Il est donc intéressant d'envisager des moyens de les surmonter.

Dans l'implémentation

L'identification des remote device : Le serveur gérant les remote device n'a pas de moyen infaillible de les identifier sur le long terme (entre plusieurs connexion-déconnexion). Les informations d'indentification sont un nom qui est attribué par le serveur (ou par l'utilisateur par l'intermédiaire du serveur) et une addresse ip. Aucune des ces deux données n'est assurément fixe après un redémarrage du device. Le moyen le plus simple de palier à ce problème serait de se baser sur un identifiant statique et unique (l'adresse MAC du module wifi d'un device par exemple).

Dans les modèles prédictifs

Types de données disponibles : Une des contraintes principale de ce projet est la limitation des types de mesure prise par "thin thermostat" (type de remote device).Cela contraint les modèle prédictif : par exemple l'utilisation des méthodes de prédiction de présence basée sur la géolocalisation des utilisateurs est bien plus précise. La température intérieure et la consommation énergétique dépendent également de bien d'autres données que de la température intérieur et extérieur et de l'ouverture d'une vanne, la luminosité, l'ouverture des porte et fenêtre et d'autres élément peuvent être pris en compte pour affiner et corrigé les modèle.

Définition du confort : En prenant une définition du confort prenant en compte plus d'élément permettrait de rendre le thermostat plus apte à satisfaire son utilisateur. Des éléments comme cité dans la sous-section 2 du chapitre 5 pourraient être insérées au thermostat.

Dans l'interaction avec l'utilisateur

Interface graphique : L'ajout d'une interface graphique permettant à l'utilisateur de pouvoir interagir plus aisément avec l'appareil, en modifiant la température visée, l'humidité, etc à tout instant sur un site internet, une application, etc.

L'accès aux informations : L'utilisateur pourrait également avoir un contrôle plus grand sur sa consommation en visualisant les heures d'ouverture de la vanne, en ayant la possibilité également de synchroniser son agenda avec le thermostat afin d'être sur que le chauffage ne se mette pas en route (Dans le cas des vacances à l'étranger par exemple).

9 Conclusion

La conception et la visualisation des demandes du projet était la première étape clé du processus de fabrication du thermostat intelligent. L'utilisation du diagramme présenté en Annexe A était le moyen le plus efficace de se représenter l'objectif final. La réalisation du câblage des différents composants a été assez rapidement effectué, tout comme l'implémentation de l'Arduino, bien que parfois celle-ci ait été retardée suite à un manque de matériel. Toutefois, certaines complications ont entraîné plusieurs modifications du câblage de l'Arduino.

L'autre partie abordée en parallèle concernait le Raspberry. La création d'un serveur était nécessaire, l'implémentation de celui-ci est telle qu'il permet une plus grande flexibilité que celle exigée par le cahier des charges. En effet, on peut y connecter n'importe quel type de Remote Device. Comme exigé dans le cahier des charges, les communications respectent le protocole HTTP.

Un stockage des données était nécessaire afin d'implémenter l'analyse intelligente. Une étude a été menée afin de choisir entre les différentes façon de faire et SQLite a finalement été retenu pour sa légèreté, sa simplicité et sa présence par défaut dans Python. L'implémentation fut sujette à une grosse modification mais respecte aujourd'hui les conventions.

L'algorithme de contrôle de la vanne a été implémenté en deux étapes, d'abord un simple « ON/OFF », suivi par après d'un algorithme intelligent et prédictif, utilisant la base de donnée pour prédire la présence future ou non de l'utilisateur. Un régulateur PID est également implémenté afin de minimiser l'oscillation de la température et de réduire la consommation énergétique.

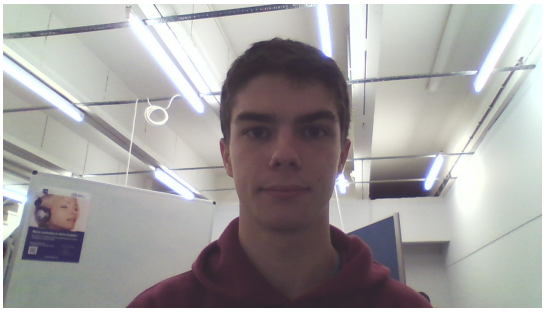
Ces algorithmes ont nécessités la mise en place de modèle mathématiques qu'il a fallu valider par la suite. Ces derniers ont donc fait l'objet de tests tout comme les thermostats individuels, centraux, etc. La plupart de ces tests furent concluants excepté pour l'actualisation des constantes du PID, il s'avère que ce processus souffrirait d'un problème de modélisation.

Le fonctionnement du groupe n'a pas nécessite de modifications durant l'année, la répartition du travail était équitable et les réunions hebdomadaires permettaient un suivi régulier du travail effectué. Cette méthode de travail semble être une bonne base pour de futurs projets.

Le travail a fournir était assez conséquent et les problèmes rencontré avec le fonctionnement ont considérablement ralenti la bonne marche du projet, mais malgré tout les quatre Milestones ont globalement été complétées. Des améliorations de ce projet sont toutefois possibles comme elles ont été énoncés plus haut.

10 Membres du groupe

Amzur Soufiane
000328725
soufiane.amz@gmail.com



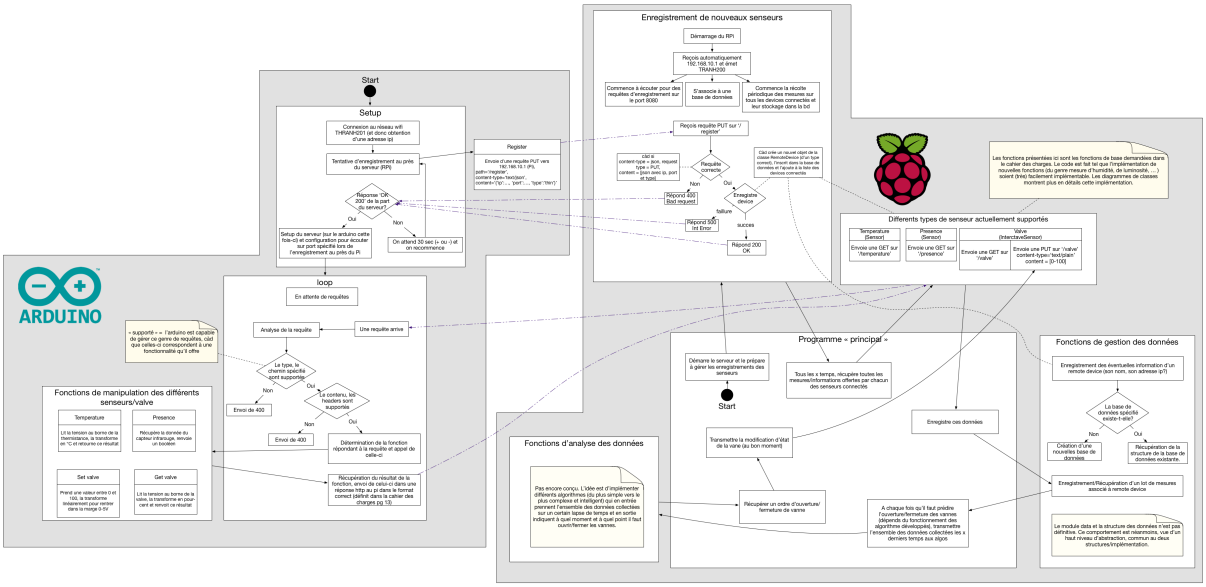
Daubry Wilson
000408780
daubry.wilson@gmail.com

Sercu Stephane
000408643
ssercu@ulb.ac.be

Verstraeten Denis
000401967
denverst@ulb.ac.be

A Diagramme de comportement global

Comme expliqué dans la partie Conception, ce diagramme a été créé principalement pour faciliter le travail de groupe. Il offre une vue très synthétique du comportement de l'ensemble du système ainsi que de la façon dont les différentes parties communiquent et interagissent. Il est également disponible sur le Git afin de pouvoir l'utiliser en zoomant dessus.



B Conventions de programmation

Dans le but de garder un code propre et maintenable par tous les participants du projet sur le long terme, des conventions et règles de bonne pratique ont été choisies pour être respectées.

1 Code Python

— les conventions proposées dans le cours de Thierry Massart [17]. Notamment pour les docstring, le nommage des classes, des fonctions et des variables,

— la langue choisie pour le code est l'anglais, pour la simple et bonne raison que c'est la langue universelle des développeurs. Le code n'en est que plus cohérent par rapport aux fonctions prédéfinies.

— L'orienté objet a été un choix facile : le code n'en est que plus compréhensible, structuré, modulaire et évolutif.

— Une attention particulière a été portée à la documentation de chaque méthode/fonction/classe pour éviter une perte de temps à la compréhension et l'analyse du code par les autres membres du groupe et/où par l'auteur lui même dans l'avenir. De cette façon, l'utilisation de ces fonctions est simplifié et ne nécessite que la lecture de la documentation associée pour savoir de quelle façon il faut les appeler et quel type de paramètre il faut leur passer.

— Les deux modules principaux (`ThermoServer` et `Data`) sont accompagnés de diagrammes de classe permettant leur documentation. Ils offrent une vue globale, simplifiée et résumée de leur structure. Cette pratique sera maintenue pour les prochains modules principalement pour son pouvoir documentaire.

2 Code Arduino

Cette partie a été programmée directement dans le code de base fournis, le style de programmation a donc été calqué sur celui-ci.

Bibliographie

[1]

[2] Narges Zaeri et Mahdi Yaghoobi Amin Mohammadbagheri. Comparison performance between pid and lqr controllers. http://www.ipcsit.com/vol7/43-S093.pdf.

[3] Arduino. Arduino - Website. https://www.arduino.cc/.

[4] Vishay BCcomponents. Ntc thermistors, radial leaded, standard precision - datasheet. https://www.vishay.com.

[5] Mohamed Belmaaziz. Cours 4 : Le confort thermique et stratégies thermiques des êtres humains. http://194.199.191.5/taiga_ftp/cours/2011/101577/Cours_thermique_du_batiment_4_confort.pdf.

[6] Jean-Paul Chehab. Interpolation polynomiale (notes de cours). http://www.lamfa.u-picardie.fr/chehab/Ens/cours_interp.pdf.

[7] Etienne Deguine et Mickaël Camus Daniel Ross. Asservissement par pid. http://lyceejdarc.org/autodoc/cours/003%20T%20STI2D/Technologie%20transversale/Asservissement/020%20Ressources%20documentairs/pid.pdf.

[8] Elec-Freaks. Specification of dyp-me003. http://elecfreaks.com/store/download/datasheet/sensor/DYP-ME003/Specification.pdf.

[9] The Python Software Foundation. HTTP servers — Python 3.5.0 documentation. https://docs.python.org/3/library/http.server.html.

[10] The Python Software Foundation. Thread-based parallelism — Python 3.5.0 documentation. https://docs.python.org/3.5/library/threading.html.

[11] Communauté francophone d'utilisateurs d'Ubuntu. Http : hypertext transfer protocol. http://doc.ubuntu-fr.org/tutoriel/console_ligne_de_commande.

[12] Emmanuel DE GEEST. Méthodes d'optimisation pour le réglage de contrôleurs pid. http://www.montefiore.ulg.ac.be/systems/degeest.pdf.

[13] Marc Haelterman. *Cours de Physique Générale, Chapitre 1, Thermodynamique*. PUB, 2014.

[14] Daniel Jackson. Http : hypertext transfer protocol. http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-170-software-studio-spring-2013/lecture-notes/MIT6_170S13_07-http-prtcol.pdf.

[15] Jeff. Raspberry Pi Thermostat Part 1 : System Overview. http://www.nooganeer.com/his/projects/homeautomation/raspberry-pi-thermostat-part-1-overview/.

[16] Araki M. Control systems, robotics, and automation. http://www.eolss.net/ebooks/Sample%20Chapters/C18/E6-43-03-03.pdf.

[17] Thierry Massart. Info-h-100 informatique - release 3.4.0 (2015) chapitre 15. http://www.ulb.ac.be/di/verif/tmassart/Informatique/latex/SyllabusPython32.pdf.

[18] Freddy Mudry. Ajustage des paramètres d'un régulateur pid. http://php.iai.heig-vd.ch/~mee/seminaires/pid_auto_ajustable/latex/napidaj.pdf.

[19] MySQL. Chapter 1 general information. http://dev.mysql.com/doc/refman/5.7/en/introduction.html.

[20] MySQL. Chapter 11 data types. http://dev.mysql.com/doc/refman/5.7/en/data-types.html.

[21] MySQL. Chapter 4 connector/python installation. https://dev.mysql.com/doc/connector-python/en/connector-python-installation.html.

[22] Artem Napov. Math-h-202 analyse numérique. http://metronu.ulb.ac.be/MATH-H-202/index.html.

[23] Raspberry Pi. Raspberry pi - website. https://www.raspberrypi.org.

[24] Raspberry Pi. Raspberry pi 2, model b. http://www.adafruit.com/pdfs/raspberrypi2modelb.pdf.

[25] Pavan Podila. Http : The protocol every web developer must know. http://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177.

[26] PostgreSQL. Chapter 8. data types. http://www.postgresql.org/docs/9.3/static/datatype.html.

[27] PostgreSQL. Postgresql 9.4.5 documentation. http://www.postgresql.org/docs/9.4/static/tutorial-arch.html.

[28] R.M. Price. RMP Lecture Notes. http://facstaff.cbu.edu/rprice/lectures/contalgo.html.

[29] Maxim Integrated Products. Arduino nano (v3.0) - user manual. http://www.jameco.com/jameco/products/prodds/2127970.pdf.

[30] PyGreSQL. Pygresql – postgresql module for python. http://www.pygresql.org/.

[31] Python. Db-api 2.0 interface for sqlite databases. https://docs.python.org/2/library/sqlite3.html.

[32] A. Kenneth Reitz. Requests : HTTP for Humans — Requests 2.8.1 documentation. http://docs.python-requests.org/en/latest/.

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

[34] SQLite. Distinctive features of sqlite. https://www.sqlite.org/different.html.

[35] Espressif Systems IOT Team. Esp8266ex datasheet. https://www.adafruit.com/images/product-files/2471/0A-ESP8266__Datasheet__EN_v4.3.pdf.

[36] Unknown. Http commands and response codes. http://papa.det.uvigo.es/~theiere/cursos/Curso_WWW/codes.html.

[37] Unknown. Linux bash shell cheat sheet. http://cli.learncodethehardway.org/bash_cheat_sheet.pdf.

[38] Unknown. Types of Controllers Proportional Integral and Derivative Controllers Electrical4u. http://www.electrical4u.com/types-of-controllers-proportional-integral-derivative-controllers/.

[39] Zennio Avance y Tecnología S.L. Advanced thermostatic control module. http://www.zennio.com/references-fr/references-fr.

[40] Karl Johan Åström. Control system design - pid control. http://www.cds.caltech.edu/~murray/courses/cds101/fa02/caltech/astrom-ch6.pdf.