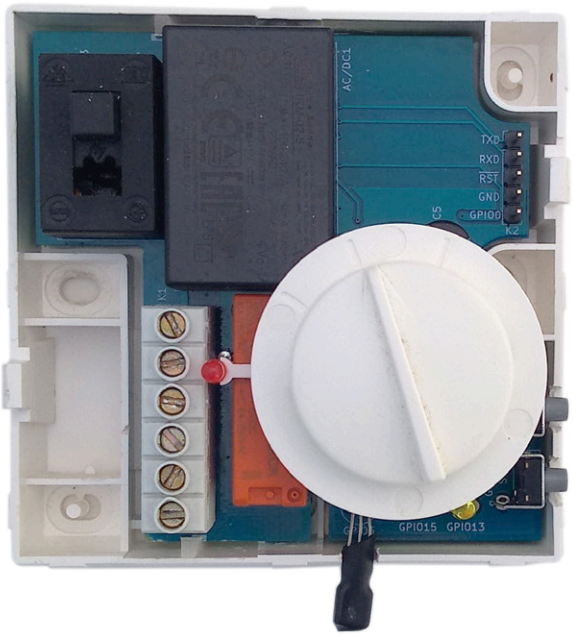
connectez votre thermostat avec ESPHome

Une tentative pour faire de la domotique comme il faut



Clemens Valens (Elektor)

La domotique bien faite est comme une main invisible. Elle vous pousse en douceur au sommet de la colline. Sans elle vous graviriez aussi cette colline, avec elle la vie est un peu plus confortable. Cet article porte sur la conception d'un thermostat pour un tel système domotique. Automatisé ou pas, son interface utilisateur classique vous permet de garder la maîtrise de la température de la pièce.

Il y a environ un an, je décidais de me lancer dans la domotique. Mon premier pas a été l'automatisation du thermostat de notre salon. Je l'ai fait en remplaçant le thermostat mural existant par le thermostat de bureau WiFi (projet Elektor 160269, publié dans l'édition de janvier/février 2018 [1][2]) reprogrammé avec un micrologiciel basé sur ESPHome. Le nouveau micrologiciel donnait accès à tous les organes du thermostat de bureau (c.-à-d. un relais, deux boutons-poussoirs et trois LED), ce qui permettait de les automatiser avec un gestionnaire domotique comme *Home Assistant*.

Dans cette configuration, le rôle du thermostat est joué par *Home Assistant* qui s'exécute sur un RPi et qui décide quand allumer ou éteindre le chauffage. Le thermostat de bureau lui-même est devenu un simple relais télécommandé avec quelques LED.

Sympa, mais...

Le système fonctionnait bien et nous a permis de passer confortablement l'hiver 2019-2020. Non sans quelques inconvénients :

- Il fallait un réseau WiFi ;
- Il fallait Home Assistant ;
- Il était moche.

Conçu pour l'avenir

Au moment de faire ces expériences, le premier problème n'en était pas vraiment un, car la plupart d'entre nous ont un réseau WiFi qui fonctionne. Lorsqu'il s'interrompt, il suffit en général de redémarrer le routeur ou son équivalent. Toutefois, on sait tous que les technologies vont et viennent, et nul ne sait si dans trente ans il y aura encore un réseau WiFi pas loin… mais nos maisons seront probablement toujours là. En d'autres termes, il nous faut une solution à l'épreuve du temps.

Compatible avec le passé

Le second point est en rapport avec le premier, car *Home Assistant* et RPi peuvent aussi disparaître un jour. Mais il y a plus. Je pense savoir comment me débrouiller avec *Home Assistant* sur un RPi, ce qui n'est pas le cas de la plupart des gens que je connais. Pour que d'autres personnes puissent utiliser mon thermostat automatisé, il doit donc être « compatible avec le passé ». Il doit ressembler à un classique thermostat mural et se comporter comme tel. L'automatisation ne doit pas être imposée, elle sera plutôt une option discrète. Elle est là pour ceux qui veulent l'utiliser, sans gêner qui n'en veut pas.

Question de goût

Le troisième problème est plutôt personnel. Mon thermostat de bureau automatisé a fini pendu au câble sortant d'un trou dans le mur où se trouvait l'ancien thermostat (**fig.1**). Quelques câbles secteur inutilisés qui sortaient du mur avaient été mis à l'abri des êtres humains stupides avec du chatterton. Et comme le thermostat de bureau a besoin d'une alimentation en 5 V avec un connecteur USB, il était relié à un chargeur de téléphone branché sur une prise multiple voisine. De sorte qu'on voyait plusieurs câbles allant vers l'appareil, là où le thermostat d'origine n'en avait pas. C'est un excellent moyen d'engager la conversation, mais la plupart des visiteurs ne trouvaient pas ça joli, même plutôt « moche », « bizarre » et « dangereux ».

Retour à la planche à dessin

Les défauts de la configuration initiale du thermostat de bureau m'ont convaincu de le reconcevoir pour résoudre certains, voire tous les inconvénients. Ceci conduisit aux spécifications suivantes :

- Réglage local de la température de consigne, c.-à-d. sur l'appareil ;
- La commande locale a priorité sur l'automatisation ;
- Alimentation par le secteur ;
- Connexions adaptées et boîtier d'aspect professionnel.

Le second item est relatif au logiciel, les trois autres concernent l'électronique du nouveau thermostat.
La spécification 1 implique une interface utilisateur intuitive pour régler la température de consigne souhaitée. Le thermostat de bureau a deux boutons qui permettent d'augmenter ou diminuer la température de consigne. En revanche, il n'y a pas d'écran pour l'afficher. Ajouter un écran avec le peu de broches d'E/S disponibles sur le module WiFi est compliqué. Utiliser un potentiomètre avec une échelle étalonnée comme le thermostat d'origine paraissait plus simple et possible grâce à l'entrée analogique du module WiFi.
La troisième spécification demande aussi réflexion. Il faut environ 1 W de puissance pour une connexion WiFi et pour alimenter le relais. On peut fabriquer une petite alimentation avec un transformateur, mais difficile de la faire suffisamment petite ; nous ne voulons pas d'une énorme boîte sur le mur. Le thermostat d'origine a une alimentation sans transformateur, ce qui est parfait pour des charges relativement constantes, mais je ne suis pas sûr que cela fonctionne bien avec une charge qui fait du WiFi. Mon intention était donc d'utiliser à la place un petit module convertisseur CA/CC.

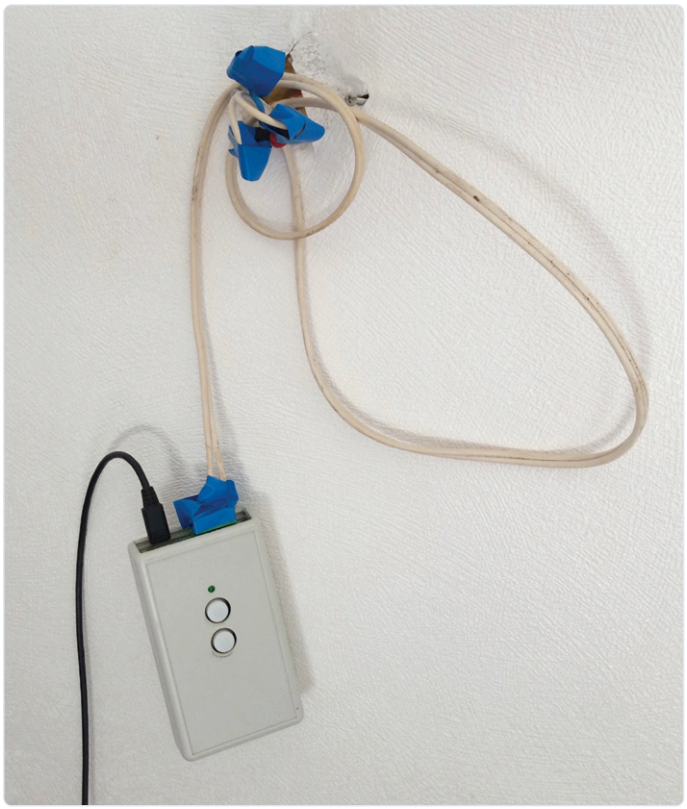


Figure 1. Certaines personnes ont trouvé « inesthétique » cette façon d'installer un thermostat intelligent dans un salon.

Je n'ai rien trouvé qui convienne en cherchant dans les catalogues de boîtiers. Fabriquer un joli boîtier sur mesure n'est pas mon fort, mais aujourd'hui avec les FabLabs, l'impression 3D et la découpe au laser accessibles même dans les endroits les plus reculés, cela pourrait ne pas être trop difficile. Je n'ai pourtant pas choisi cette voie. J'ai préféré essayer de réutiliser le boîtier du thermostat d'origine qui avait déjà tout ce que je voulais : un potentiomètre avec une échelle, une LED et un interrupteur. De plus, il avait des trous de montage bien placés et un bon moyen de connecter un circuit imprimé au secteur en traversant le fond pour cacher tous les câbles.
Mon effort de conception se réduisait maintenant à tasser le circuit du thermostat de bureau redessiné sur un circuit imprimé de sorte qu'il puisse contenir dans le boîtier existant avec le potentiomètre, la LED, l'interrupteur d'alimentation et la connexion au secteur, dans exactement les mêmes positions que dans le thermostat d'origine.
Reconcevoir le circuit du thermostat de bureau fut assez simple (**fig.2**). J'ai remplacé l'alimentation USB par un module 5 V CA/CC, et ajouté un potentiomètre avec une résistance de limitation de tension, car

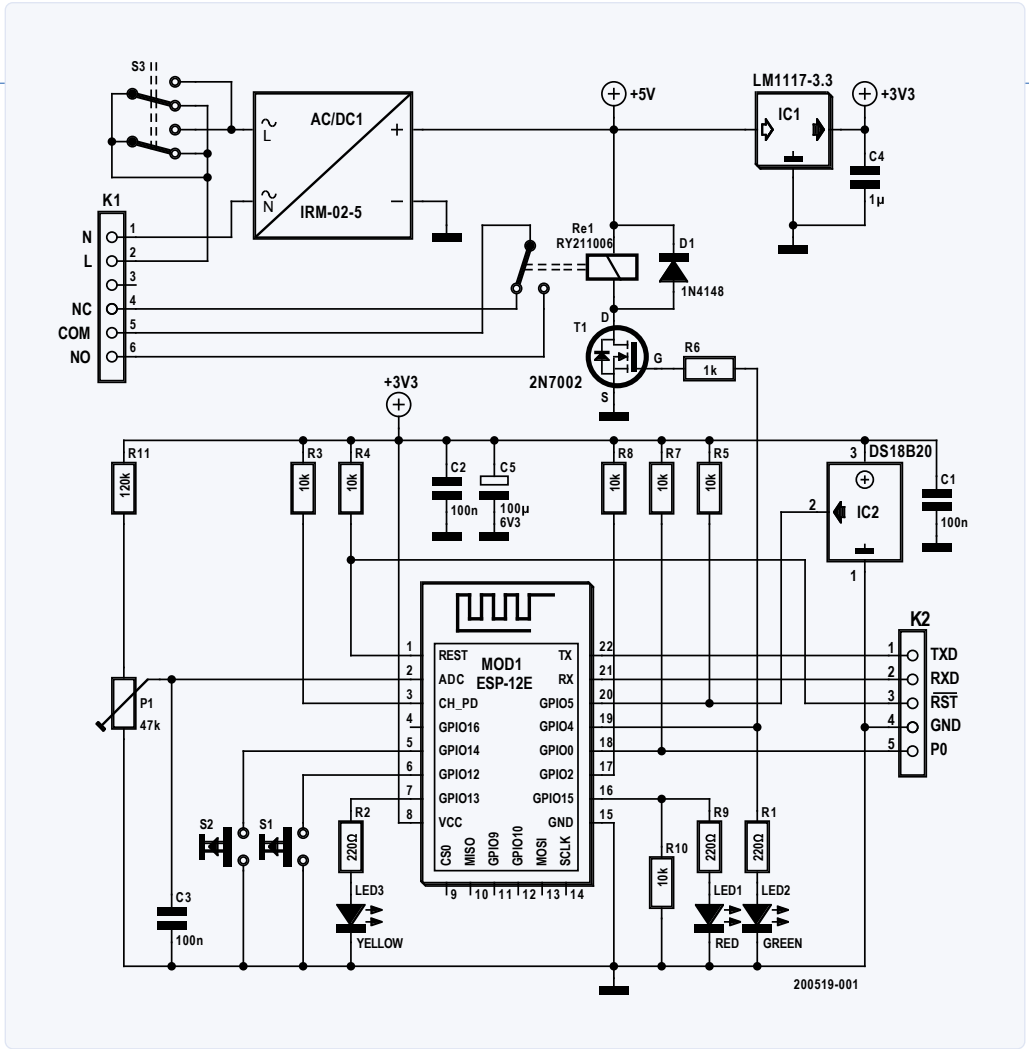


Figure 2. À la base, le nouveau thermostat discrètement connecté est le Thermostat de Bureau d'Elektor de 2018 avec une nouvelle alimentation. Un potentiomètre règle la température de consigne.

le module WiFi ne peut pas gérer de tensions supérieures à 1,1 V. J'ai gardé les deux boutons-poussoirs et les trois LED qui pourraient être utiles tôt ou tard.

Par une heureuse coïncidence, le module CA/CC s'est révélé juste assez petit pour tenir sous le support du bouton en plastique du potentiomètre. Le potentiomètre, l'interrupteur et le bornier du secteur ont été récupérés sur le vieux thermostat (**fig.3**). Restait à remplacer le relais 48 V par un modèle de 5 V. Quelle chance, une fois de plus, le relais du vieux thermostat était d'un type issu d'une famille au standard industriel toujours disponible et qui existe en 5 V !

Il a fallu beaucoup mesurer pour tout mettre sur un circuit imprimé qui s'adapte au boîtier d'origine, mais ça y est. Tous les CMS y compris le module WiFi sont sous la carte (**fig.4**), tandis que tous les composants traversants sont dessus. Une petite surface de carte supplémentaire a été gagnée en obstruant une paire de trous inutilisés du boîtier. Je confesse que je n'ai pu router toutes les pistes que par une approche inévitablement assouplie des normes d'isolation.

Logiciel

Je devais aussi repenser le micrologiciel ESPHome que j'avais compilé pour ma première configuration. Au lieu de juste donner accès à tous les capteurs et organes de commande du thermostat et laisser *Home Assistant* s'occuper du reste, je devais maintenant réaliser les automatismes dans ESPHome. Programmation et configuration sont faites dans le fichier YAML du projet ESPHome du thermostat (voir « la domotique c'est facile avec… » [3]).

Mesurer la température de la pièce

Déclarez d'abord le capteur de température qui mesure la température de la pièce. Comme le capteur utilisé est un DS18B20 (de Dallas à l'origine, maintenant de Maxim ou même Analog Device) connecté à la broche 5 du GPIO, et comme ESPHome a un composant spécifique pour Dallas, cela se traduit ainsi :

```
dallas:
  - pin: GPIO5

sensor:
 - platform: dallas
   address: 0x6D00000C24013928
   name: "Measured temperature"
   id: t_room
   filters:
    - offset: 0.0
```

La première ligne indique à ESPHome d'inclure son module de communication Dallas 1-wire et d'y connecter le GPIO5. Le capteur est ensuite issu de la plateforme *dallas*. L'*adresse* est optionnelle. Toutefois, si vous la spécifiez, elle doit être correcte ; vous pouvez l'obtenir à partir du journal d'ESPHome (n'utilisez pas la mienne, elle est unique). Ici il faut spécifier un *id* (*t_room*), car nous devrons nous référer au capteur depuis un autre endroit dans ce fichier YAML (cf. ci-dessous). L'ajout de filtres permet si nécessaire d'effectuer certaines corrections sur la température mesurée.

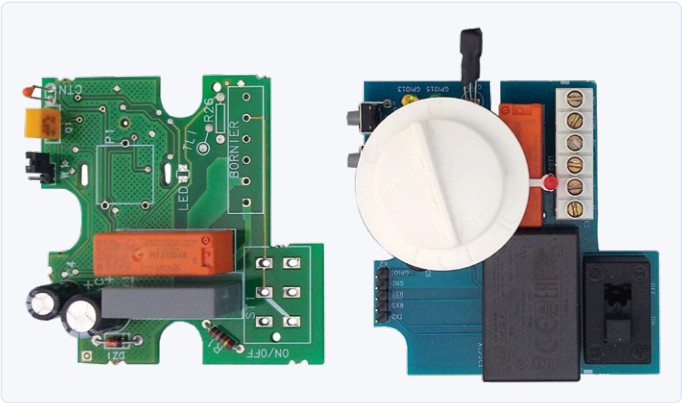


Figure 3. Le nouveau projet réutilise quelques morceaux de l'ancien thermostat, notamment le boîtier, ce qui explique la forme particulière du nouveau circuit imprimé. D'où résultent aussi quelques contraintes sur la position de certains composants.

Figure 5. *Après quelques ajustements mineurs du boîtier de l'ancien thermostat (un peu de plastique coupé), la nouvelle carte s'adapte parfaitement.*

Température de consigne

On règle la température de consigne avec le potentiomètre. Comme la valeur est une tension, ajoutez-le à la section ***sensor*** du fichier YAML avec plateforme ***adc*** comme type de capteur. Les « ***sensor filters*** » permettent de convertir la tension d'entrée en une température qui corresponde à l'échelle imprimée sur le boîtier. L'équation $T_{target} = 25 \cdot V_{in} + 6{,}75$ convient assez bien dans mon cas. Ceci se traduit par un filtre ***multiply*** avec une valeur de 25 et un filtre ***offset*** avec une valeur de 6,75. En spécifiant les unités en °C, un gestionnaire domotique comme *Home Assistant* traitera la donnée en tant que température.

```
- platform: adc
   name: "Target temperature"
   id: t_target
   icon: "mdi:temperature-celsius"
   pin: A0
   update_interval: 5s
   # Convert potentiometer scale to °C (min=6.75°C,
  max=31.75°C)
   filters:
    - multiply: 25.0
    - offset: 6.75
```

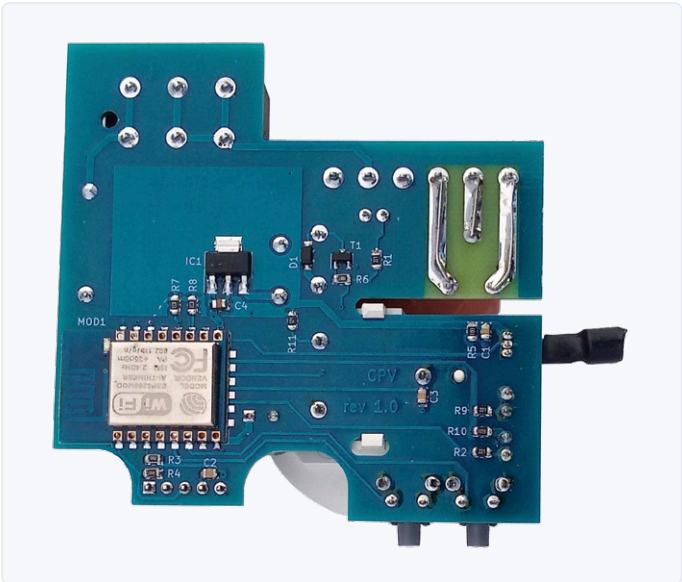


Figure 4. Tous les CMS sont sous la carte. Les pistes connectées au chauffage (en haut à droite) ont été renforcées, avec plus de soudure, pour leur permettre de conduire plus de courant sans surchauffer. La découpe juste en dessous d'elles fait office d'isolation galvanique entre les parties sous haute et basse tension. La protubérance noire à droite est le capteur de température, protégé par une gaine thermorétractable. Tout ce qui produit de la chaleur est à gauche de la carte qui est censée pointer vers le haut lorsque le thermostat est en place contre le mur.

Figure 6. Assemblage terminé. Les deux boutons-poussoirs sont accessibles par deux découpes (à gauche) et le capteur de température dépasse en dessous. Les LED verte et jaune ne sont visibles qu'à travers les fentes d'aération pour éviter d'éclairer la pièce la nuit. Elles sont de couleur différente pour qu'on voie bien laquelle des deux est allumée.

```
    unit_of_measurement: "°C"
    on_value:
     then:
     - lambda: |-
       auto call = id(t_controller).make_call();
       call.set_target_temperature(x);
       call.perform();
```

La partie à la fin après *on_value:* est un automatisme et sera expliquée après l'introduction du composant « climate » ci-dessous. Mais d'abord nous examinerons le relais.

Commuter le chauffage

C'est très simple, car le relais est juste un interrupteur connecté à la broche 4 du GPIO et fait donc partie de la plateforme *gpio*. Comme le capteur de température, il faut un *id (heater)* pour le rendre accessible depuis les autres sections du fichier YAML.

```
switch:
   - platform: gpio
     pin: GPIO4
     name: "Heater"
     id: heater
```

Climatisation

Bon, nous avons maintenant un capteur pour mesurer la température de la pièce (*t_room*), un potentiomètre pour régler la température de consigne (*t_target*) et un relais pour commuter le chauffage (*heater*).

ESPHome dispose d'un composant *climate* pour commander les appareils de chauffage et de rafraichissement. Un thermostat est donc un composant *climate*. L'avantage d'utiliser un composant intégré est de vous économiser du travail. On a aussi la garantie d'avoir un joli composant graphique de commande dans l'interface utilisateur de *Home Assistant*.

```
climate:
   - platform: thermostat
     name: "Thermostat"
     id: t_controller
     sensor: t_room
     default_target_temperature_low: 20 °C
     heat_action:
      - switch.turn_on: heater
     idle_action:
      - switch.turn_off: heater
     hysteresis: 0.5
     away_config:
      default_target_temperature_low: 15 °C
```

Notre thermostat ne rafraîchit pas ; il se contente de chauffer. Cela fait de lui un membre de la plateforme *thermostat* d'ESPHome. Il a besoin d'un *id* pour pouvoir être commandé par d'autres choses dans le fichier YAML. J'ai mis l'ID à *t_controller* parce que la plateforme s'appelle déjà *thermostat* et qu'il ne faut pas tout mélanger.
Le composant *climate* dispose d'une entrée capteur de température, que nous connectons à *t_room*. Il doit aussi être connecté à un chauffage, ce que nous pouvons faire grâce au commutateur *heater* défini plus tôt.

Comme moi, vous vous attendriez à ce qu'un composant *climate* ait aussi une entrée pour la température de consigne, mais il n'en a pas. (Peut-être sera-t-elle ajoutée dans une prochaine version d'ESPHome ?). À la place, il a une commande intégrée de réglage de la température de consigne, un peu comme le potentiomètre du thermostat. Heureusement, on peut contourner cette limitation par l'utilisation de blocs nommés « lambda ».

Blocs Lambda

Le concept est à la fois remarquable et horrible. Remarquables, car ils vous laissent faire tout ce que vous voulez ou presque, et affreux parce qu'ils défient tous les principes de configuration d'un appareil par le biais d'un simple fichier YAML sans avoir à connaître le C++.
En clair, un bloc lambda est du code C++ injecté tel quel dans le projet ESPHome. Pour pouvoir faire cela, les développeurs ont dû se démener pour parvenir à une interface utilisable entre YAML et C++. Le résultat est un code lambda plus compliqué qu'il ne l'aurait été s'il s'était agi de simple C++.
En fait, dès que vous vous retrouverez à utiliser les blocs lambda, vous devriez commencer à penser à la création d'un composant personnalisé. ESPHome prend en charge les composants personnalisés pour quasiment tout. J'étais sur le point d'écrire un composant *climate* personnalisé pour résoudre mes problèmes quand j'ai décidé de garder ce sujet pour un autre article.
Retour au réglage de la température de consigne du composant *climate*. Son interface C++ a une fonction pour fixer la température de consigne. Le bloc lambda de la section d'automatisation *on_value* du capteur potentiomètre précédent (reproduite ci-dessous) montre comment l'utiliser. Chaque fois qu'une nouvelle valeur est disponible, la méthode *set_target_temperature* du composant *climate t_controller* est appelée, comme ceci :

```
  on_value:
     then:
     - lambda: |-
       auto call = id(t_controller).make_call();
       call.set_target_temperature(x);
       call.perform();
```

En C++ normal, cela donnerait quelque chose comme cela :

```
t_controller.set_target_temperature(x);
```

Si le composant *climate* avait eu une entrée de température de consigne, les choses auraient été encore plus simples. Par exemple :

```
climate:
   - platform: thermostat
     name: "Thermostat"
     id: t_controller
     sensor: t_room
     target: t_target
     …
```

Passons.
Malheureusement, cette façon d'automatiser *on_value* est trop simpliste, elle outrepasse la commande à distance de la température de consigne (par ex. avec *Home Assis*tant). Pour résoudre cela, l'automatisation ne devrait pouvoir fonctionner que lorsque t_target change (c.-à-d. lorsque quelqu'un tourne le potentiomètre). Comme ESPHome n'a pas

d'automatisation *on_value_changed* pour les capteurs, nous pouvons gérer cela nous-mêmes dans le bloc lambda, comme ceci :

```
on_value:
   then:
   - lambda: |-
     static float x_last = 0.0;
     if (x<x_last-0.1 || x>x_last+0.1)
     {
      x_last = x;
      auto call = id(t_controller).make_call();
      call.set_target_temperature(x);
      call.perform();
     }
```

La faible hystérésis sur x (±0,1 dans ce cas) améliore l'immunité de l'automatisme au bruit du capteur.

Le filtre delta

Une autre solution – plus élégante – consiste à utiliser le filtre delta sur *t_target*. Ce filtre ne transmet une nouvelle valeur que si elle diffère de la précédente de plus ou moins delta. Ainsi, si delta = 1 et que la dernière valeur était égale à 20, alors la prochaine valeur ne sera transmise que si elle est soit inférieure à 19 soit supérieure à 21.

```
- platform: adc
   name: "Target temperature"
   id: t_target
   …
   filters:
    - multiply: 25.0
    - offset: 6.75
    - delta: 0.2
   …
```

Les filtres sont exécutés dans l'ordre où ils apparaissent, c.-à-d. que le filtre delta travaille sur la valeur convertie en degrés Celsius et pas directement sur la tension en entrée. Sa valeur doit être faible, pour pouvoir facilement ajuster un peu le thermostat vers le haut ou vers le bas, ce qui fait toute la différence pour le confort de l'utilisateur.

Terminer le dispositif

Une fois le fichier YAML prêt, on peut télécharger le micrologiciel sur le module WiFi. La première fois (c.-à-d. avec un module qui ne fonctionne pas déjà avec un logiciel compatible avec ESPHome), on doit utiliser le port série pour cela (disponible sur le connecteur K2). Retrouvez la procédure exacte en [3]. Une fois que le dispositif fonctionne avec ESPHome et que la programmation *Over-the-Air* (OTA) est activée (lorsque le fichier YAML contient la ligne *ota*:), on peut reprogrammer le thermostat sans être physiquement connecté au système de développement. En d'autres termes, on peut l'installer à la place du thermostat existant.
La disposition du thermostat est telle que le capteur de température n'est pas réchauffé par le module WiFi ni l'alimentation.

Conclusion

Cet article a montré comment on peut remplacer un banal thermostat par un autre, fait-maison, intelligent et connecté, compatible avec la domotique. Bricoler un prototype qui fasse à peu près ce qu'il est censé faire n'a pas été trop difficile. Ce qui demande plus d'efforts, c'est de le rendre utilisable par tout le monde, à tout moment, et sans malmener l'esthétique. Nul doute qu'il reste à peaufiner certains réglages du nouveau thermostat, mais comme je pourrai le reprogrammer à distance (sur le terrain comme on dit), ce sera facile.

200519-02

Votre avis, s'il vous plaît ?
Veuillez adresser vos questions et vos commentaires par courriel à clemens.valens@elektor.com ou à redaction@elektor.fr.

Ont contribué à cet article

Auteur : **Clemens Valens**
Schémas : **Patrick Wielders**
Rédaction : **C. J. Abate**
Maquette : **Giel Dols**
Traduction : **Denis Lafourcade**

PRODUITS

- **ESP-12F, ESP8266-based Wi-Fi module**
  www.elektor.fr/esp-12f-esp8266-based-wi-fi-module-160100-92
- **Thermostat de bureau WiFi – circuit imprimé nu (160269-1)**
  www.elektor.fr/wi-fi-desktop-thermostat-bare-pcb-160269-1
- **Carte à microcontrôleur NodeMCU ESP8266**
  www.elektor.fr/nodemcu-microcontroller-board-with-esp8266-and-lua
- **H. Henrik Skovgaard, *IoT Home Hacks with ESP8266* (Elektor, 2020)**
  www.elektor.fr/iot-home-hacks-with-esp8266

LIENS

[1] **Roy Aarts & Clemens Valens, " thermostat de bureau wifi", Elektor, janv.-fév. 2018** : www.elektormagazine.fr/magazine/elektor-201801/41300
[2] **Clemens Valens, «Wi-Fi Thermostat at Elektor Labs,» Elektor Labs, 2018** : http://bit.ly/wifi-thermostat
[3] **Clemens Valens, «la domotique, c'est facile avec…», sept.-oct. 2020** : www.elektormagazine.com/200019-02