le projet Panettone

système de gestion et de maintien d'un ferment au levain

Daniel Fantin (Australie)

Ce projet Arduino est né de la frustration de beaucoup, beaucoup de Panettone ratés. Pour mémoire, le panettone est une brioche italienne traditionnelle préparée pour Pâques et Noël. Elle a une texture incroyablement légère et aérée, qui nécessite une levure très active provenant d'un levain…

Figure 1. Un magnifique Panettone, n'est-ce pas ?

Jetez un coup d'œil à la **figure 1**. Voilà à quoi ressemble un Panettone réussi : léger, aéré, gorgé de beurre et de fruits. Si vous avez l'eau à la bouche, il y a ici tout ce qu'il faut (sur le plan technique) pour préparer parfaitement vous-même un aussi magnifique gâteau. *Buon appetito !*

Qu'est-ce qu'un ferment au levain ?

En termes simples, il s'agit d'une association de bactéries et de levures qui, nourries de farine fraîche et d'eau, libèrent des gaz qui créent un pain aéré et levé. La levure doit être suffisamment forte pour lutter contre la charge gravitationnelle du pain ou de la pâte à Panettone. Dans le cas du pain, ce n'est pas si terrible, mais pour le Panettone, nous nous heurtons à de grandes quantités de beurre, de fruits et de sucre – une tâche difficile pour toute levure.

Quand un levain est-il assez fort pour être utilisé dans un Panettone ? C'est simple : Quand il triple de hauteur en quatre heures, tout en étant maintenu à 27 °C. Cela paraît simple, pour le moins, mais ce maintien est difficile. Personne n'a vraiment envie de chauffer sa maison à 27 °C juste pour son levain, et encore moins de maintenir cette température pendant des semaines tandis que la force du levain augmente. Et comment savoir s'il a triplé de volume en quatre heures ? Allez-vous vérifier en permanence ? Programmer des alarmes et garder un mètre ruban dans la cuisine ? C'est trop, trop difficile, trop accaparant. Il vaut mieux laisser ça à un robot.

Que se passe-t-il lorsque le levain n'est pas assez fort ? Le Panettone qui en résulte est dense, difficile à manger et semble raté. Qu'en est-il lorsqu'il est assez fort ? Revenez à la **figure 1**. Vous savez juste qu'il est délicat, moelleux, aéré, velouté, sucré, croustillant, beurré… tout ce que vous en attendez.

Nous sommes donc au mois de mai au moment où j'écris ces lignes, et j'ai du temps avant de devoir faire ma prochaine fournée de Panettone (en décembre). Lorsque, dans ma discipline à l'université de Deakin, on nous a demandé de résoudre un problème du « monde réel », Pâques venait de passer, et je venais juste d'échouer avec un tas d'autres Panettone. Cela semblait donc être un problème du monde réel qui valait la peine d'être résolu.

Tout d'abord, j'ai pensé à acheter une chambre de pousse ou quelque chose de similaire, mais ce sont des articles CHERS, surtout si l'on considère que ce que j'avais l'intention de construire avait encore plus de fonctionnalités (et était, au global, moins cher).

Critères du projet

Qu'est-ce que le projet doit faire ?

1. Gérer la température du ferment au levain.
2. Nous dire s'il est assez fort pour le Panettone.

Donc, il me faut :

- quelque chose pour suivre la hauteur du levain à l'intérieur du bocal (cela me permettra de savoir si le levain a triplé de hauteur en quatre heures)
- quelque chose pour suivre la température du levain dans le bocal
- quelque chose pour chauffer ou refroidir le bocal afin qu'il reste toujours à environ 27 °C

- quelque chose pour m'avertir des événements importants (par exemple, le seuil des quatre heures)
- une interface graphique pour pouvoir réinitialiser le ferment de levain après l'avoir nourri
- une interface graphique pour pouvoir suivre à distance les variables clés (température, croissance)

Je voulais utiliser les principes de conception basés sur les composants afin de pouvoir toujours isoler les problèmes à un seul endroit. Alors que ce projet aurait facilement pu être regroupé sur une seule carte, je voulais créer quelque chose qui puisse tester l'interface entre différents systèmes, et m'instruire pour entreprendre des projets plus compliqués à l'avenir. C'est pourquoi j'ai opté pour :

- **Capteurs :** un RPi 3B+ pour gérer le DHT et les capteurs de distance (à base de laser) et transmettre ces données sans fil (vers la carte Particle, voir ci-dessous) et via USB série (vers l'Arduino).
- **Chauffage/refroidissement :** un Arduino UNO pour allumer les plaques chauffantes et le ventilateur lorsque nécessaire, afin de maintenir la température du levain.
- **GUI/notifications/remontées :** une carte Particle Argon [1] pour donner à l'utilisateur une interface graphique à distance, une interface graphique LCD, et alerter sur les événements importants.

La **figure 2** présente un schéma rapide de l'architecture du système.

Webhooks/IFTTT

Tout devrait pouvoir communiquer avec le reste. Je n'avais pas beaucoup d'expérience dans ce domaine, et j'ai pensé que la plateforme IFTTT [2] (*If This Then That* : si ceci, alors cela) offrait le moyen le plus simple et le plus facile de mettre en place cette communication. J'ai donc créé un compte IFTTT, puis j'ai généré une applet avec un webhook comme « IF ». En gros, **SI** le *webhook* reçoit une requête web avec une charge utile JSON (**J**avaScript **O**bject **N**otation) (c'est-à-dire une requête du RPi), **ALORS** il déclenche une fonction Particle. Celle-ci peut alors utiliser les données JSON comme elle le doit.
Le site Web PiMyLifeUp [3] m'a beaucoup aidé. Le plus important est que le *webhook* vous donne une adresse de la forme :

```
https://maker.ifttt.com/trigger/
    tempsent/json/with/key/XXXXXX
```

Où XXXXXX est votre clé personnelle. Il ne nous reste plus qu'à utiliser cette URL dans notre code pour déclencher cette applet en la postant au moment opportun.
Le deuxième IFTTT est une connexion entre la Particle et le téléphone – c'est assez simple. L'**IF** attend que la Particle publie un événement d'un nom spécifique, puis envoie une notification à l'application IFTTT sur le téléphone si et quand elle est reçue.

Le Raspberry Pi

Maintenant, faisons en sorte que tous les capteurs fonctionnent, et que le RPi communique avec tout. Pour connecter le circuit :

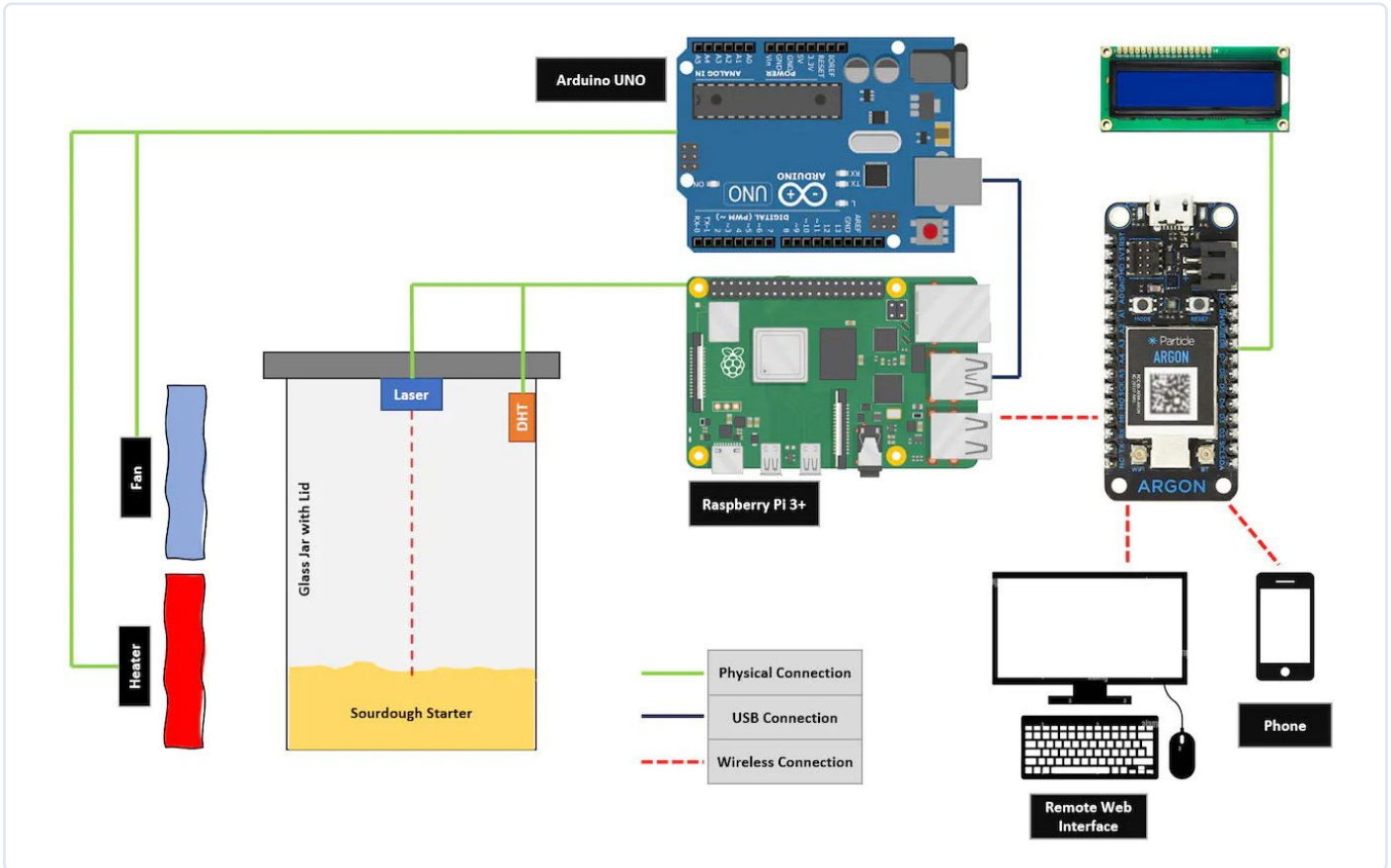

Figure 2. Schéma de l'architecture du système.

- **DHT** : Connectez la broche d'alimentation du DHT au 5 V du RPi, la deuxième broche au GPIO4 du RPi, et la quatrième broche au GND du RPi. La troisième broche sur le capteur DHT est inutilisée. Nota : Il aurait été préférable d'utiliser un DHT22 pour la précision, mais le mien était défectueux et je l'ai remplacé par un DHT11 pour le produit final.
- **VL53L1XLaser** : Connectez Vin au 3,3 V du RPI, GND au GND du RPi, SDA au GPIO2 du RPi, et SCL au GPIO3 du RPi. Ceci permet la communication série entre le RPi et le Laser.

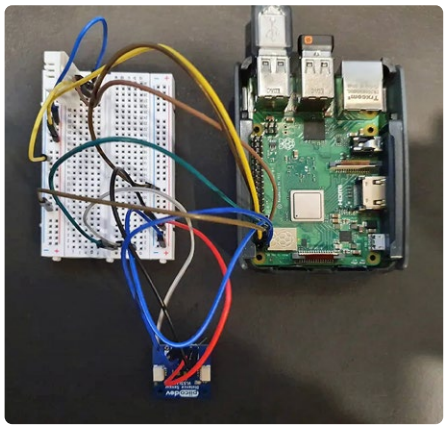

Figure 3. Montage de test pour les modules laser et température.

À noter ici que j'ai choisi un laser pour mesurer la distance, plutôt qu'un capteur à ultrasons, pour quelques raisons :

- Précision – les mesures au laser sont beaucoup plus précises.
- Un laser est plus fiable, d'autant plus que la surface de la pâte est souple.
- Je n'aime pas ennuyer mes chats avec du bruit ultrasonique permanent toute la journée.

La **figure 3** montre ma configuration de test pour les modules de mesure de distance et de température par laser ; les connexions doivent être effectuées comme sur la **figure 4**. Les noms des ports sont indiqués sur le schéma pour améliorer la visibilité du câblage. À noter que le schéma ne montre pas la connexion USB entre l'Arduino et le RPi, alors qu'il s'agit d'un composant essentiel pour la communication.

Maintenant, place au code. J'ai créé un nouveau fichier Python sur le RPi et commencé l'édition. J'ai utilisé l'EDI Python, Thonny [3], car je le trouve très facile à utiliser. Commençons par les importations :

- **Adafruit DHT** [5] pour l'utilisation du capteur DHT
- **PiicoDev-VL53L1X** de Core Electronics [6] pour le laser
- **Time/Sleep**
- **Requests** pour les webhooks et la communication sans fil
- **Serial** pour la communication via USB avec l'Arduino

Quelques éléments de configuration

On utilise des drapeaux pour signaler à la Particle que le ventilateur ou le chauffage sont

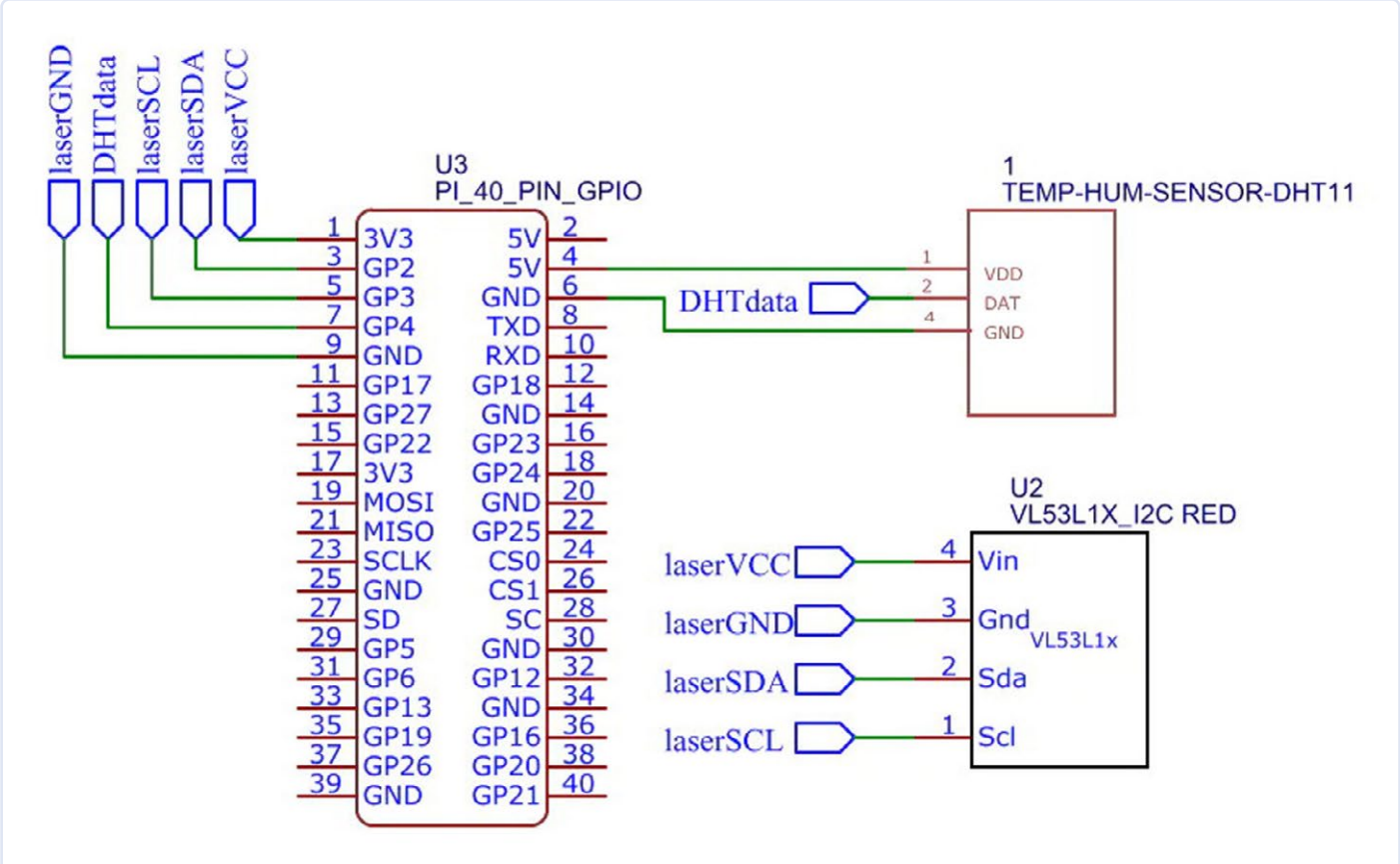

Figure 4. Les connexions réalisées à la figure 3.

en marche. L'« airmessage » est transmis en série comme un entier à l'Arduino pour « faire quelque chose ». Le port série a été configuré à l'adresse correcte en la vérifiant dans le terminal : `ls /dev/tty*`

J'ai créé une méthode de « lectures moyennées » pour la robustesse – elle prend cinq lectures consécutives et en fait la moyenne avant de les envoyer aux webhooks/série, etc. Cela permet d'éviter que d'uniques lectures erronées ne déclenchent des événements par erreur. Il y a également une vérification du code de gestion des erreurs qui envoie à l'utilisateur une notification si 100 lectures erronées sont observées.

Il y a de nombreux commentaires dans la boucle principale, principalement utilisés à des fins de débogage. Cette boucle recueille la moyenne et décide ensuite de ce qu'il faut faire :

- Si la température est inférieure à 24 °C, mettre le chauffage en marche.
- Laisser la température monter jusqu'à 27 °C, puis éteindre le chauffage.
- Si la température est > 30 °C, mettre le ventilateur en marche.
- Laissez la température descendre à 27, puis éteindre le ventilateur.
- Tant que la température est comprise entre 24 et 30 °C (la zone optimale pour le levain), aucun dispositif n'est actif.

Il en résulte le script Python *PiCodepy*, qui fait partie du logiciel et peut être téléchargé gratuitement sur le site Web de l'article [7].

Arduino

La configuration de l'Arduino est plus difficile du point de vue de l'alimentation électrique, mais heureusement, mon père est un expert en électronique. Le ventilateur a besoin de 12 V, ce qui est bien plus que ce que peut fournir l'Arduino. De plus, les plaques chauffantes prennent 5 V mais ont une résistance de 6 Ω. Le courant nécessaire est donc proche de 1 A - bien plus que ce qu'une broche Arduino peut supporter. Par conséquent, c'est une source d'alimentation externe qui fournit la puissance nécessaire au ventilateur/chauffage lors du déclenchement par l'Arduino.

La configuration :

- **Chauffage :** Connectez la broche 12 de l'Arduino à un côté de la bobine du relais, et l'autre côté au GND de l'Arduino. Connectez le +5 V de l'alimentation au rail commun du relais. Connectez le pôle positif des plaques chauffantes du côté NO du relais.
- **Ventilateur :** Connectez la broche 10 de l'Arduino à un côté de la bobine du relais, et l'autre côté au GND de l'Arduino. Connectez le +12 V de l'alimentation au rail commun du relais. Connectez le pôle positif du ventilateur du côté NO du relais.
- **GND :** Connectez le GND des plaques chauffantes et des ventilateurs au GND de l'alimentation.
- **Arduino :** Connectez une batterie de 9 V (ou une alimentation électrique) à l'Arduino. Seul l'Arduino est alimenté par la batterie, PAS le RPi.

La **figure 5** montre le montage expérimental et la **figure 6** le circuit du système autour du chauffage et du ventilateur. L'Arduino reçoit des données du RPi via USB (série). Les messages sont codés en UTF-8. Tout ce que nous devons faire est d'obtenir le code série et d'exécuter l'action appropriée en fonction

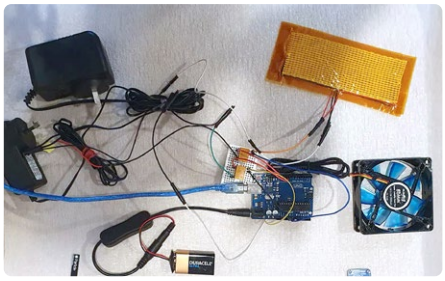

Figure 5. Montage vite fait du système.

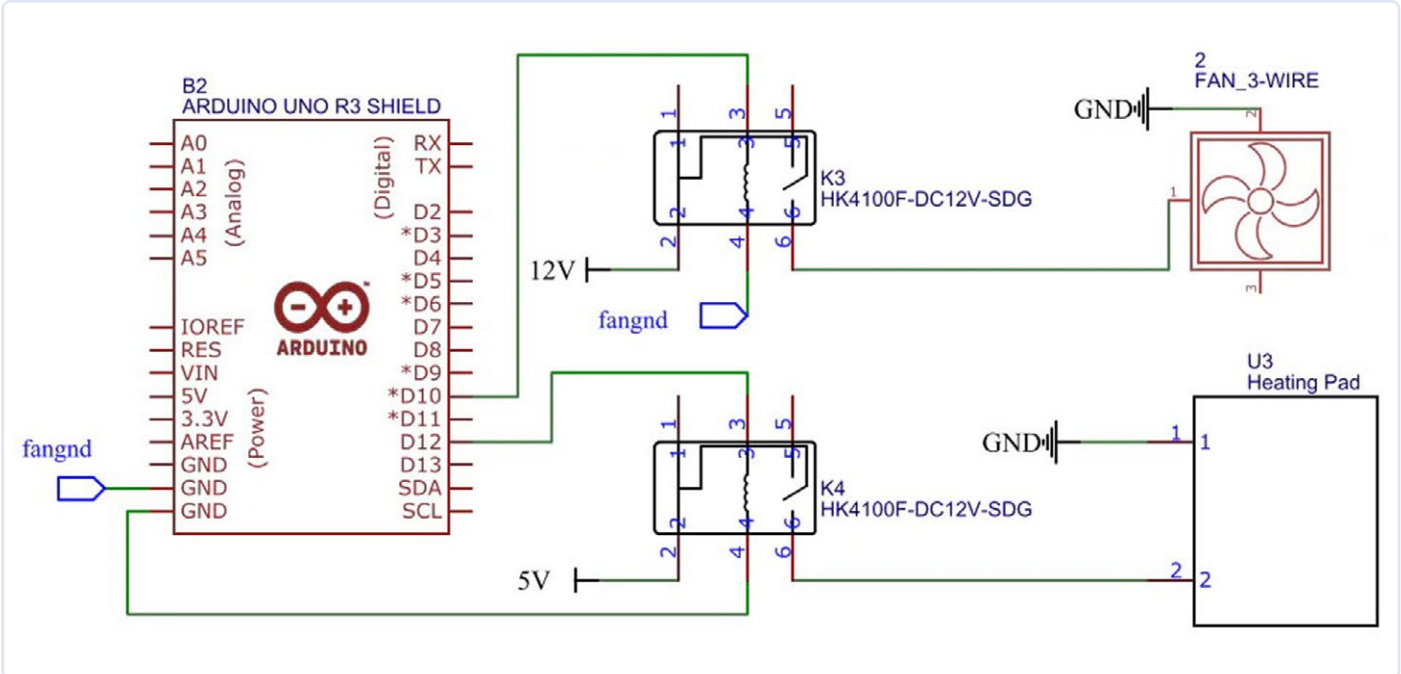

Figure 6. Connexions autour de la carte Arduino.

de ce code. Ceci est géré par une instruction if dans la boucle principale du code ci-dessous. Le site Web d'Automatic Addison [8] a été très utile à cet égard. Le code Arduino se trouve dans un fichier à l'adresse [7].

Particle Argon

La Particle Argon est au centre de la communication entre l'utilisateur, le code et les dispositifs. Le LCD, le potentiomètre et les LED doivent être connectés à la carte. Le LCD peut utiliser l'I²C, mais j'ai dû connecter quelques lignes en plus (**figure 7**). Le site Web Hackster [9] m'a aidé à bien le faire.

J'ai ensuite connecté deux LED : Une verte pour « le levain est bon ! » et une rouge pour « le levain n'est pas bon ». Elles ont été connectées aux broches D6 et D7 de l'Argon, puis mises à la masse via une résistance de 220 Ω. Après avoir ajouté le potentiomètre, le sous-système Particle était prêt (**figure 8**). Le schéma complet se trouve à la **figure 9**. Le code de la Particle [7], généré dans l'EDI Particle, est le plus long et le plus compliqué, car il relie tout. Parcourrons-le :

- Création de l'objet `lcd`, effacement des chaînes, initialisations nécessaires.
- Codage en dur des constantes de la hauteur initiale du pot et de la hauteur de départ de la pâte, car il s'agit de valeurs connues et fixes.
- Création de la fonction `resetbutton()`. Celle-ci est activée à distance via HTML pour réinitialiser le minuteur une fois que les quatre heures sont écoulées (et que les résultats sont clairs).

L'initialisation

- Effacer le LCD et afficher un message d'introduction.
- Initialisation des deux fonctions qui interagissent avec IFTTT. displayTemp

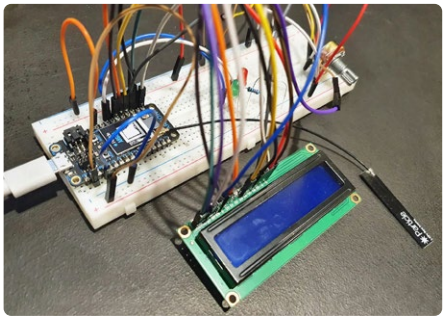

Figure 8. Configuration de la carte Particle.

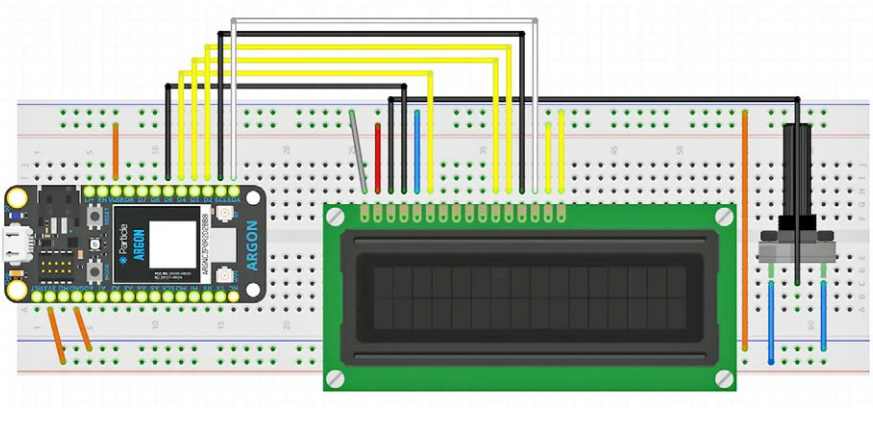

Figure 7. L'écran et la carte Particle.

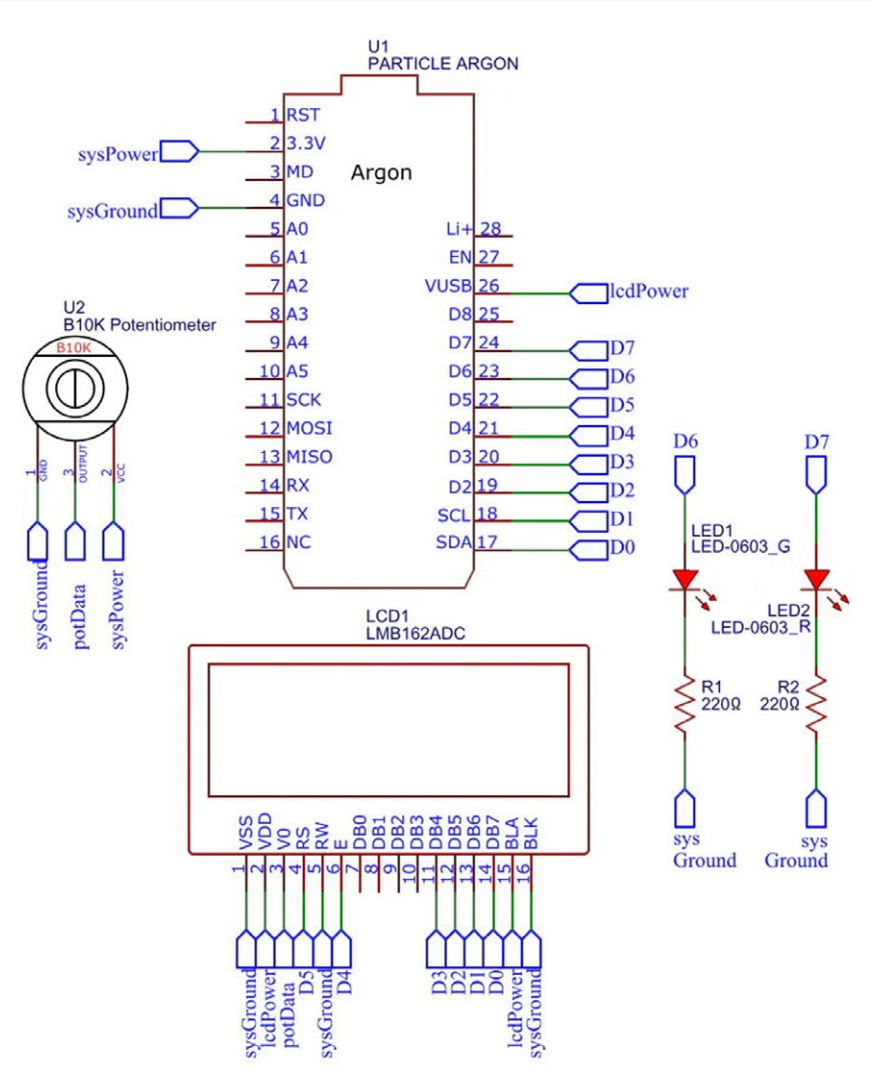

Figure 9. Schéma de la carte Particle et de ses composants.

exécute la fonction `tempDisplay()` et reset exécute la fonction qui réinitialise le bouton.
- Enregistrement de l'heure de départ.
- Extinction des LED.

La boucle
- Il y a un délai de dix secondes pour que le LCD ne soit pas surchargé par de continuels rafraichissements.
- Impression des commandes courantes (cmd1 et cmd2). Celles-ci sont définies dans la fonction `tempDisplay()`.
- Calcul de la croissance du levain.
- Si la minuterie dépasse quatre heures, un nouveau message indiquant le résultat final apparaît à l'écran.

La fonction tempDisplay()

C'est là que sont reçues les données JSON provenant du *webhook* mentionné précédemment. Ce sont toutes les données critiques de distance du RPi, transmises sans fil à la Particle. Malheureusement, je n'ai pas réussi à composer le JSON pour qu'il soit facilement accessible. Au lieu de cela, je l'ai simplement pris comme une chaîne de caractères que j'ai décomposée en sous-chaînes, ce qui était très simple. Je me suis assuré que les règles de formatage de RPi étaient strictes afin que cela fonctionne toujours correctement. Le JSON contenait les données suivantes : Distance, Température, et si le ventilateur ou le chauffage était allumé. Ces données ont été utilisées pour gérer à la fois le HTML et le LCD. Ce code peut également être trouvé sur [7].

HTML

On a utilisé l'HTML pour permettre l'accès à distance au système. Cela me permet de réinitialiser la minuterie lorsque les quatre heures sont écoulées, ainsi que de surveiller la température et la croissance depuis n'importe où avec un accès à l'Internet.

Le code est trop long pour être imprimé ici, mais nous pouvons montrer quelques fonctions. L'essentiel du code est tiré du tutoriel Particle [10]. Les fonctions clés du HTML étaient :
- Afficher les données de température en direct (en s'abonnant à un flux d'événements).
- Afficher les données de croissance en direct (en s'abonnant à un flux d'événements).

Listage 1. Les appels à particle.getEventStream()

```
particle.getEventStream({ name: 'tempEvent', auth: sessionStorage.
  particleToken }).then(
{
    function (stream) {
        console.log('starting event stream');
        stream.on('event', function (eventData)
        showTemp(eventData)
    });
});

particle.getEventStream({ name: 'growEvent',auth: sessionStorage.
  particleToken }).then( function (stream)
{
   {
    console.log('starting event stream');
    stream.on('event', function (eventData)
    showGrow(eventData)
   });
});

particle.getEventStream({ name: 'timeEvent', auth:sessionStorage.
  particleToken }).then(
    function (stream) {
    {
       console.log('starting event stream');
       stream.on('event', function (eventData)
       showTime(eventData)
    });
});

Listing resetControl
function resetControl(cmd) {
     //const deviceId = $('#deviceSelect').val();
    $('#statusSpan').text('');
    particle.callFunction({ deviceId: 'X', name: 'reset', argument:
  cmd,auth:sessionStorage.particleToken}).then(err);
    function (data) {
        $('#statusSpan').text('Reset completed'); },
    function (err) {
         $('#statusSpan').text('Error calling device: ' +
    } );
}
```

Listage 2. Fonction resetControl()

```
function resetControl(cmd) {
 //const deviceId = $('#deviceSelect').val();
    $('#statusSpan').text('');
    particle.callFunction({ deviceId: 'X', name: 'reset', argument:
  cmd, auth:sessionStorage.particleToken}).then(err);
    function (data) {
        $('#statusSpan').text('Reset completed'); },
    function (err) {
         $('#statusSpan').text('Error calling device: ' +
} );
}
```

- Remise à zéro de la minuterie (en appelant une fonction lorsqu'on appuie dessus).

Les appels `particle.getEventStream()` (**listage 1**) montrent comment les données sont extraites du flux d'événements. La fonction `resetControl()` (**listage 2**) est déclenchée par l'appui sur un bouton (l'ID de la Particle a été supprimé).

Vues en action

La **figure 10** montre quelques photos du LCD à différents moments. La dernière partie, d, montre la notification de c sur un smartphone. La **figure 11** montre le système complet du robot (a), la boîte avec l'électronique à l'intérieur (b) et les cartes et fils sur le dessus de la grande boîte (c). Et, si vous vous demandez à quoi va ressembler ce levain, la **figure 12** montre la croissance féroce du levain entre avant et après la session de quatre heures.

Conclusion

C'est tout. Tout fonctionne, et tout est synchronisé. Maintenant, il est temps de produire automatiquement le premier ferment au levain avec ce robot à pâte. Et, bien sûr, quand le levain est assez fort, faites cuire et mangez !

220416-04 — VF : Denis Lafourcade

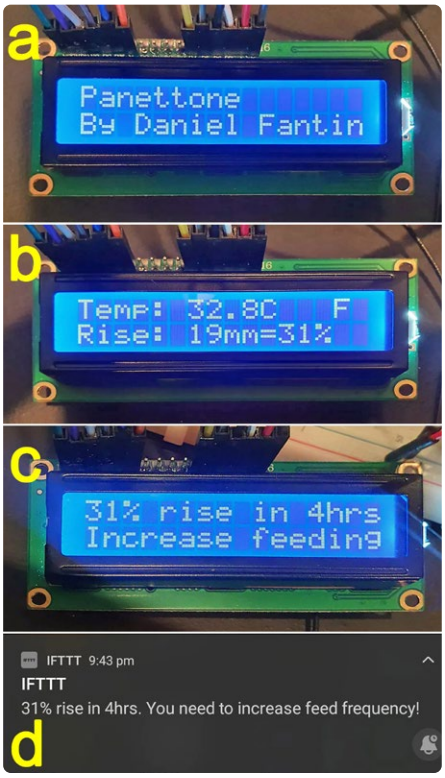

Figure 10. Différents contenus sur le LCD (a, b et c), plus le message du smartphone après quatre heures.

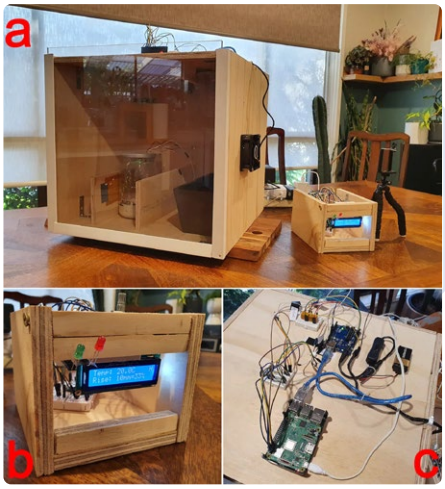

Figure 11. Le robot complet (a) avec le boîtier de commande (b) et quelques cartes et fils (c).

Figure 12. Le ferment au levain avant (a) et après avoir atteint la hauteur correcte quatre heures plus tard (b).

À propos de l'auteur

Daniel Fantin est étudiant en deuxième année d'informatique à l'université de Deakin. Il est passionné par la nourriture et la technologie, alors combiner la cuisine italienne et la robotique est un prolongement logique.

Des questions, des commentaires ?

Si vous avez des questions techniques, n'hésitez pas à contacter l'auteur par l'intermédiaire de Hackster [12] ou l'équipe de rédaction d'Elektor à l'adresse redaction@elektor.fr.

LIENS

[1] Particle Argon : https://docs.particle.io/argon/
[2] IFTTT : https://ifttt.com
[3] PiMyLifeUp : https://pimylifeup.com/using-ifttt-with-the-raspberry-pi/
[4] EDI Thonny : https://thonny.org
[5] DHT Adafruit : https://github.com/adafruit/Adafruit_Python_DHT
[6] PiicoDev_VL53L1X : https://elektor.link/piicodevgithub
[7] Logiciel du projet : www.elektormagazine.fr/220416-04
[8] Automatic Addison : https://elektor.link/rpiarduino2way
[9] Site web de Hackster : https://elektor.link/particlehackster
[10] Tutoriel sur la Particle : https://elektor.link/particleapitut
[11] Projet sur hackster.io : https://elektor.link/hacksterpanettone
[12] Daniel Fantin à hackster.io : www.hackster.io/danielfantin