UNIVERSITÉ CÔTE D'AZUR

POLYTECH NICE-SOPHIA

ELSE4 FISA

MÉMOIRE DE PROJET

Régulation du CO_2 dans une ruche

Réalisé par :
M. CARNEVALI

Encadré par :
M. PETER

Année universitaire :
2024 - 2025
15 mai 2025

CARNEVALI Nicolas
ELSE4

Table des matières

CARNEVALI Nicolas
ELSE4

Introduction :

La préservation de la santé des colonies d'abeilles constitue aujourd'hui un enjeu majeur tant sur le plan écologique qu'économique. En effet, les abeilles jouent un rôle essentiel dans les écosystèmes par leur action de pollinisation, contribuant à la reproduction de nombreuses espèces végétales et à la sécurité alimentaire mondiale. Cependant, depuis plusieurs décennies, les apiculteurs sont confrontés à de multiples menaces compromettant la survie des ruches. Parmi celles-ci, l'infestation par les **Varroa**, un acarien parasite, représente l'un des facteurs les plus préoccupants.

Le varroa est un ectoparasite redoutable qui s'attaque principalement aux abeilles adultes et aux larves en se nourrissant de leur corps gras. Cette spoliation énergétique fragilise considérablement les abeilles, réduisant leur espérance de vie et altérant leur système immunitaire. De plus, la présence de ce parasite perturbe profondément le comportement des butineuses, affectant notamment leurs capacités d'apprentissage, d'orientation et de retour à la ruche. Un phénomène aggravant est observé lorsque certaines abeilles infestées, désorientées, se trompent de ruche, contribuant ainsi au phénomène de dérive et favorisant la dissémination du parasite au sein d'autres colonies voisines.

Face à ce fléau, les méthodes de lutte traditionnelles montrent aujourd'hui leurs limites, notamment en raison du développement de résistances chez le varroa et des effets secondaires sur les colonies d'abeilles. Dans ce contexte, notre projet s'inscrit dans une démarche d'innovation en explorant des approches alternatives de contrôle du varroa fondées sur les différences physiologiques entre le parasite et son hôte. Nos études préliminaires ont mis en évidence que les varroas présentent une plus grande sensibilité au dioxyde de carbone (CO_2) que les abeilles, suggérant ainsi une piste de régulation sélective du parasite au sein des ruches.

Ainsi, l'objectif principal de ce travail est d'étudier l'impact de l'injection contrôlée de CO_2 dans les ruches comme méthode de lutte ciblée contre les varroas, tout en veillant à préserver la santé et le comportement normal des abeilles. Pour ce faire, nous avons analysé les tolérances respectives des deux espèces à différentes concentrations de CO_2, en prenant en compte les paramètres physiologiques et comportementaux. Ce mémoire présente les résultats de nos expérimentations, ainsi que les perspectives qu'offre cette méthode innovante pour une apiculture durable et respectueuse de l'environnement.

CARNEVALI Nicolas
ELSE4

J’ai pu avoir le plaisir de travailler sur ce projet lors de deux années de suite, ce mémoire comportera donc mes expériences sur ce projet lors de ces deux dernières années.

Travaux menés lors de l’année 2023/2024 :

Lors de ma première année, j’ai pu travailler en binôme sur ce projet avec mon camarade M. ZANTOUR. Nous avons pu avoir la chance de reprendre ce projet à un stade avancé car un autre groupe avait déjà étudié ce sujet. Ainsi, nous étions en possession d’une carte électronique, de schéma électronique ainsi que de certains codes de tests.

Nous avons donc entrepris des tests sur une carte préalablement conçue par un groupe d’une année d’étude précédente. Cette carte, contrôlée par un ESP32, est équipée de trois capteurs de CO2 de référence CCS811. L'objectif de cette carte est de mesurer le niveau de CO2 à différents emplacements dans la ruche en disposant la carte verticalement. Cette disposition est cruciale étant donné que la densité de l'air peut entraîner des variations significatives du taux de CO2 à différents niveaux de la ruche.

Lors des tests initiaux, nous avons rencontrés des difficultés avec les capteurs de CO2 (CCS811). Les mesures obtenues n'étaient pas satisfaisantes, ce qui a nécessité des améliorations. C'est ainsi que nous avons décidé de développer une deuxième version de la carte électronique, intégrant de nouveaux types de capteurs, plus précis.

Tout d'abord, l'ajout d'un ENS160, ce module est capable de mesurer non seulement le taux de CO_2, mais également la concentration totale de composés organiques volatils (TVOC) ainsi que la qualité générale de l'air. Nous avons donc regardé les modules comportant un ENS160 et trouvé une combinaison d’un ENS160 et d’un ATH20. Ce module, permet donc de mesurer le taux de CO_2 ainsi que la température et l’hygrométrie. Ces derniers nous accorderons donc des informations supplémentaires qui ne sont pas négligeable sachant que le taux de CO_2 peut varier en fonction de la température.

L'un des avantages majeurs de ces modules est leur facilité d'intégration avec l'ESP32. Chaque module peut utiliser soit le protocole I2C, soit le SPI pour communiquer avec l'ESP32, offrant ainsi une flexibilité dans la configuration. De plus, chaque module est équipé de huit broches, comprenant principalement des broches d'alimentation et de communication, ce qui simplifie grandement le processus de connexion et d'intégration dans le système global.

Pin d’alimentation :

- VIN – 3.3V, il s'agit des broches d'alimentation, les capteurs de gaz et de température ayant une tension d’alimentation de 3.3V, nous avons donc inclus un régulateur de tension à bord qui convertira 5VDC en 3.3VDC. Ces deux bornes sont pontées.
- GND - Masse commune pour l'alimentation et la logique.

Pin de communication :

- CS : Le Chip Select permet de définir le protocole de communication du module. S’il est au niveau logique HAUT, alors l’I2C est le protocole actif. Sinon, le protocole utilisé est le SPI. Si vous changez de I2C à SPI alors que le module est alimenté alors il est impossible de rebasculer en I2C sans éteindre le composant et mettre la pin CS au niveau HAUT.
- MOSI/SDA : MOSI est utilisé en mode SPI c’est Master Output Slave Input ; en mode I2C c’est la pin permettant de transmettre les données dans les deux sens (Master/Slave et Slave/Master).
- SCLK/SCL : pin transmettant l’horloge peut importe le mode.
    - MISO/ADDR : MISO est utilisé en SPI, c’est Master Input Slave Output. Lors de l’utilisation de l’I2C, cela permet de changer l’adresse. 0x53 lorsque que le niveau logique bas est actif et 0x52 lors que le niveau logique est haut.

CARNEVALI Nicolas
ELSE4

Pin d'interruption :

- INIT : Pin d'interruption (non utilisée)

Figure 1 - Le nouveau module utilisé (ENS160+AHT20)

Sur cette version de la carte, nous avons rencontré d'importants défis lors de la détection et de l'intégration des capteurs. Ces capteurs utilisent à la fois les protocoles de communication I2C et SPI, ce qui ajoute une complexité supplémentaire à leur utilisation. Une broche cruciale de connexion, appelée "CS", est utilisée pour définir le mode de communication du capteur : lorsque cette broche est maintenue à un niveau bas, le capteur fonctionne en mode SPI, tandis que lorsqu'elle est à un niveau haut, le capteur communique en mode I2C. De plus, chaque capteur possède une adresse I2C qui doit être unique pour permettre une communication correcte avec le microcontrôleur.
Cependant, nous avons constaté une limitation significative : ces capteurs ne permettent que deux adresses I2C différentes, généralement 0X52 ou 0X53, en fonction du niveau logique de la broche ADDR. Cette contrainte a posé un défi majeur lors de l'intégration des capteurs, car nous avions besoin de trois mesures simultanées pour obtenir des données précises sur le niveau de CO2, la qualité de l'air, le TVOC et la température.

Malgré ces défis, nous avons réussi à obtenir des mesures cohérentes en utilisant la bibliothèque Arduino Adafruit ENS160 pour tester les capteurs de CO2, la qualité de l'air et le TVOC. Cependant, la limitation des adresses I2C nous a confrontés à une contrainte majeure : nous n'avons pu obtenir que deux mesures simultanées sur les deux capteurs ENS160 disponibles. Une mesure était obtenue à partir de l'adresse 0X52 et une autre à partir de l'adresse 0X53. Cela signifie que nous ne pouvions pas récupérer les données du troisième capteur de CO2 ni obtenir simultanément la mesure de température avec le taux de CO2.
Pour la mesure de la température, nous avons utilisé la bibliothèque DFRobot_AHT20. Dans notre quête pour obtenir des mesures complètes, nous avons envisagé une solution alternative. L'idée était d'effectuer deux mesures simultanées avec deux capteurs, puis d'éteindre l'un des capteurs et d'allumer le troisième pour obtenir la troisième mesure. Cependant, cette approche a rencontré des obstacles. Le principal défi résidait dans le temps de préchauffage nécessaire pour obtenir des mesures précises. Ce temps était d'au moins 60 secondes, ce qui rendait cette solution inefficace pour une acquisition de données en temps réel.
Un peu de détails sur les bibliothèques utilise pour effectuer les mesures :

Adafruit ENS160 : Cette bibliothèque est développée par Adafruit et est spécifiquement conçue pour les capteurs environnementaux. Elle prend en charge la communication avec des capteurs tels que le CCS811 et l'ENS160. Avec cette bibliothèque, vous pouvez obtenir des mesures de CO2, de TVOC (composés organiques volatils totaux) et de qua- lité de l'air. Elle offre une interface conviviale et des fonctions faciles à utiliser pour récupérer les données des capteurs. De plus, elle est bien documentée, ce qui facilite son utilisation pour les développeurs.

Lien : https://www.arduino.cc/reference/en/libraries/ens160-adafruit-fork/

DFRobot_AHT20 : Cette bibliothèque est développée par DFRobot et est spécifiquement conçue pour le capteur de température et d'humidité AHT20. Elle permet de récupérer facilement les mesures de température et d'humidité à partir du capteur AHT20. Comme la bibliothèque Adafruit ENS160, elle offre une interface simple et des fonctions prêtes à l'emploi pour interagir avec le capteur. Elle est également bien documentée, ce qui facilite son intégration dans les projets Arduino.

Lien : https://github.com/DFRobot/DFRobot_AHT20

```
16:47:03.834 -> Concentration of total volatile organic compounds : 21 ppb
16:47:03.834 -> Carbon dioxide equivalent concentration : 400 ppm
16:47:03.834 ->
16:47:04.815 -> Sensor operating status : 2
16:47:04.815 -> Air quality index : 2
16:47:04.815 -> Concentration of total volatile organic compounds : 69 ppb
16:47:04.860 -> Carbon dioxide equivalent concentration : 494 ppm
16:47:04.860 ->
16:47:05.857 -> Sensor operating status : 2
16:47:05.857 -> Air quality index : 4
16:47:05.857 -> Concentration of total volatile organic compounds : 1028 ppb
16:47:05.857 -> Carbon dioxide equivalent concentration : 1009 ppm
16:47:05.857 ->
16:47:06.873 -> Sensor operating status : 2
16:47:06.873 -> Air quality index : 3
16:47:06.873 -> Concentration of total volatile organic compounds : 437 ppb
16:47:06.873 -> Carbon dioxide equivalent concentration : 847 ppm
16:47:06.873 ->
16:47:07.846 -> Sensor operating status : 2
16:47:07.885 -> Air quality index : 3
16:47:07.885 -> Concentration of total volatile organic compounds : 328 ppb
16:47:07.885 -> Carbon dioxide equivalent concentration : 811 ppm
16:47:07.885 ->
16:47:08.869 -> Sensor operating status : 2
16:47:08.869 -> Air quality index : 3
16:47:08.869 -> Concentration of total volatile organic compounds : 272 ppb
16:47:08.869 -> Carbon dioxide equivalent concentration : 775 ppm
16:47:08.915 ->
```

Figure 2 - Exemple de mesures du CO2 + qualité de l'air + TVOC

Dans notre processus d'amélioration du système, nous avons entrepris la conception d'une troisième version, intégrant non seulement les nouveaux capteurs, mais également un commutateur multiplexeur I2C de référence PCA9548A. Cette évolution visait à résoudre le défi rencontré précédemment, celui de gérer efficacement la communication avec plusieurs capteurs à l'aide d'un unique microcontrôleur ESP32.

Le PCA9548A est un composant essentiel dans cette conception. Il s'agit d'un commutateur multiplexeur I2C offrant huit canaux distincts. Son rôle principal est de permettre l'extension des capacités de communication I2C d'un microcontrôleur en autorisant la connexion simultanée de plusieurs périphériques sur un même bus I2C. L'aspect crucial de ce commutateur réside dans sa capacité à sélectionner dynamiquement l'un des huit canaux disponibles, et ainsi à diriger le flux de données I2C vers le périphérique connecté à ce canal spécifique.

Concrètement, cela signifie que chaque capteur peut être connecté à l'un des huit canaux du PCA9548A. Ensuite, l'ESP32 peut sélectionner le canal correspondant au capteur dont il souhaite récupérer les données, en écrivant simplement les commandes appropriées via le bus I2C. Cette approche offre une solution élégante pour gérer plusieurs capteurs sur un seul bus I2C, en permettant au microcontrôleur de communiquer sélectivement avec chaque capteur selon les besoins du système.

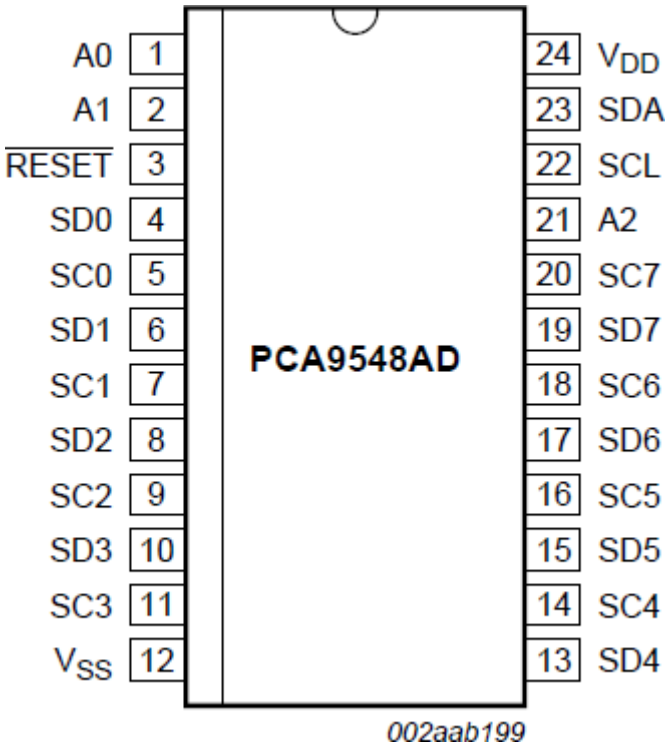


Figure 3 : MUX PCA9548A

Pour commencer le processus de connexion, nous avons pris soin de connecter la broche d'alimentation du PCA9548A, ainsi que les broches SDA (données), SCL (horloge) et RST (réinitialisation) au niveau bas, assurant ainsi un état initial stable pour le commutateur multiplexeur. Ensuite, nous avons établi les connexions entre les broches de commande (A0, A1 et A2) du PCA9548A et les broches correspondantes de l'ESP32, permettant à ce dernier de contrôler sélectivement les canaux du MUX. Une fois la configuration de base effectuée, nous avons procédé à la connexion des broches de sortie du MUX (SD0, SC0) au premier capteur. Cette étape consistait à acheminer les signaux de données (SD) et d'horloge (SC) du canal sélectionné vers les broches correspondantes du capteur. Ce processus a été répété pour les capteurs suivants, avec les broches SD1, SC1 et SD2, SC2 connectées aux capteurs 2 et 3 respectivement. Chaque étape de connexion a été soigneusement réalisée en suivant les spécifications du schéma de câblage et en vérifiant la conformité avec les besoins de communication des capteurs. Cette approche méthodique garantit une configuration fiable et cohérente du système, assurant ainsi un fonctionnement optimal lors de l'acquisition de données par l'ESP32 à partir des différents capteurs.

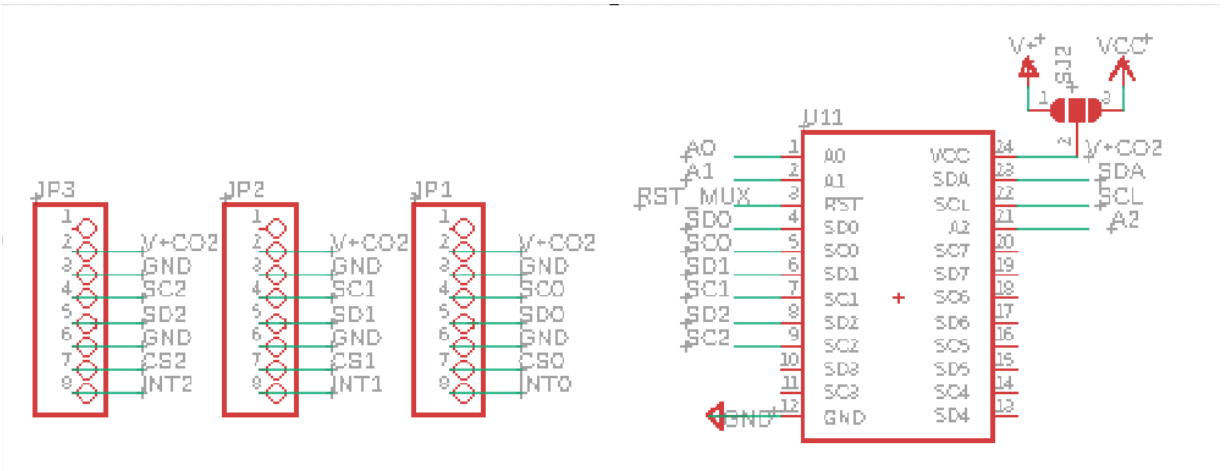


Figure 4 : Nouveau schématique pour le MUX et les 3 capteurs

Après avoir terminé les connexions et faire le schématique des composants essentiels, nous avons entamé le processus de routage de la carte. Cette phase cruciale du développement consiste à concevoir les chemins de connexion sur le circuit imprimé afin d'assurer une communication efficace et fiable entre tous les composants. "Cette phase du développement a été particulièrement exigeante et a demandé un investissement important en termes de temps et d'apprentissage. En effet, nous avons dû nous familiariser avec le logiciel de conception de cir- cuits imprimés (CAD) Eagle, tout en apprenant les tenants et aboutissants de la conception de circuits imprimés.

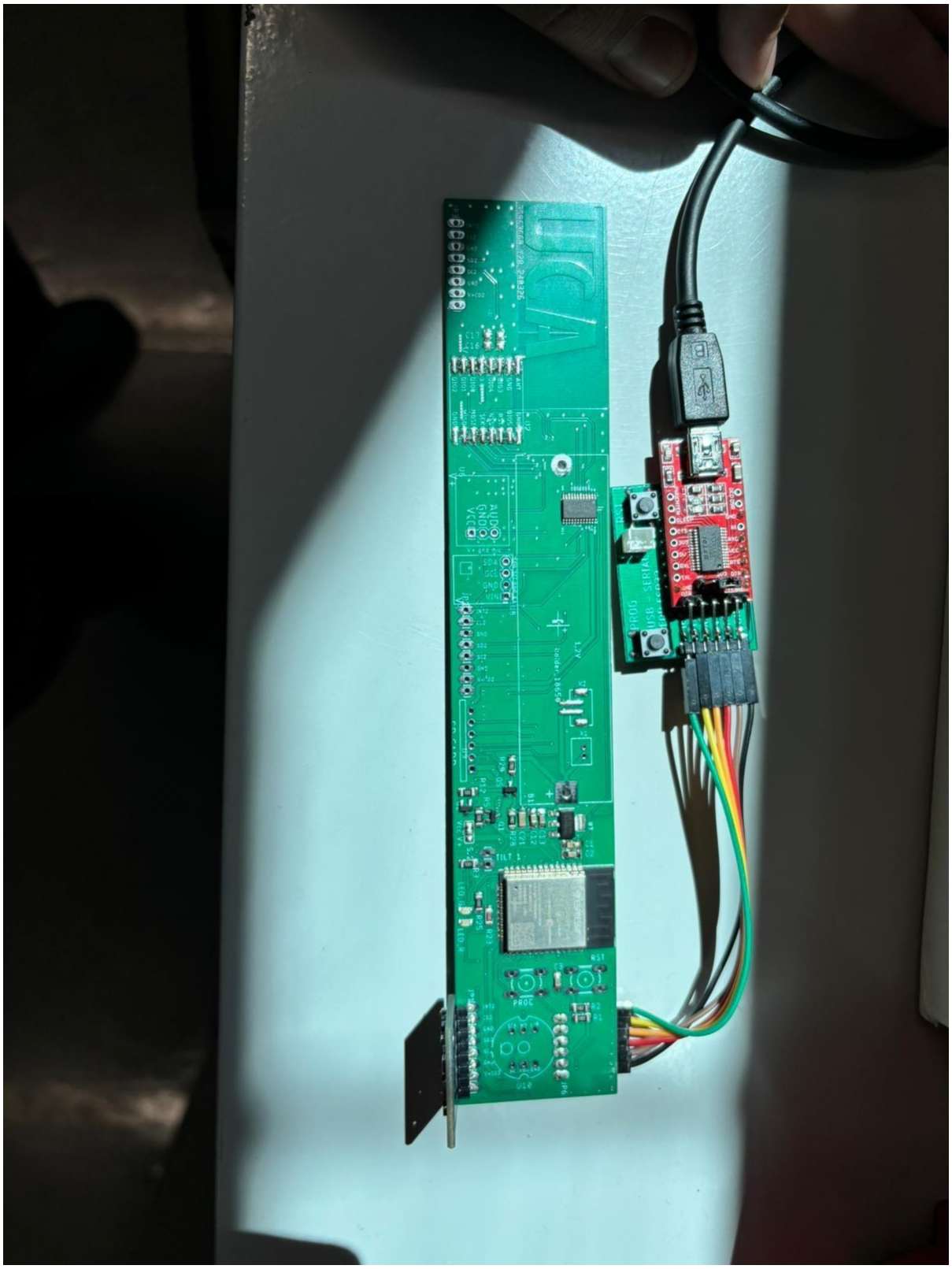
Nous avons également dû nous conformer à des normes rigoureuses, telles que les espacements entre les vias et les pistes, qui jouent un rôle crucial dans la fiabilité et les performances du circuit imprimé. Par exemple, nous avons appris que les espaces entre les vias et les pistes doivent respecter des dimensions spécifiques, avec un espacement de 12 pour les connexions et de 32 pour les pistes d'alimentation, conformément aux normes de l'industrie.

Ce processus d'apprentissage et de mise en pratique des normes de conception a été essentiel pour garantir la qualité et la robustesse de notre circuit imprimé final. Bien que cela ait repré- senté un défi, cela nous a également permis d'acquérir de nouvelles compétences et une exper- tise précieuse dans le domaine de la conception de circuits imprimés, renforçant ainsi notre capacité à réaliser des projets futurs avec succès.

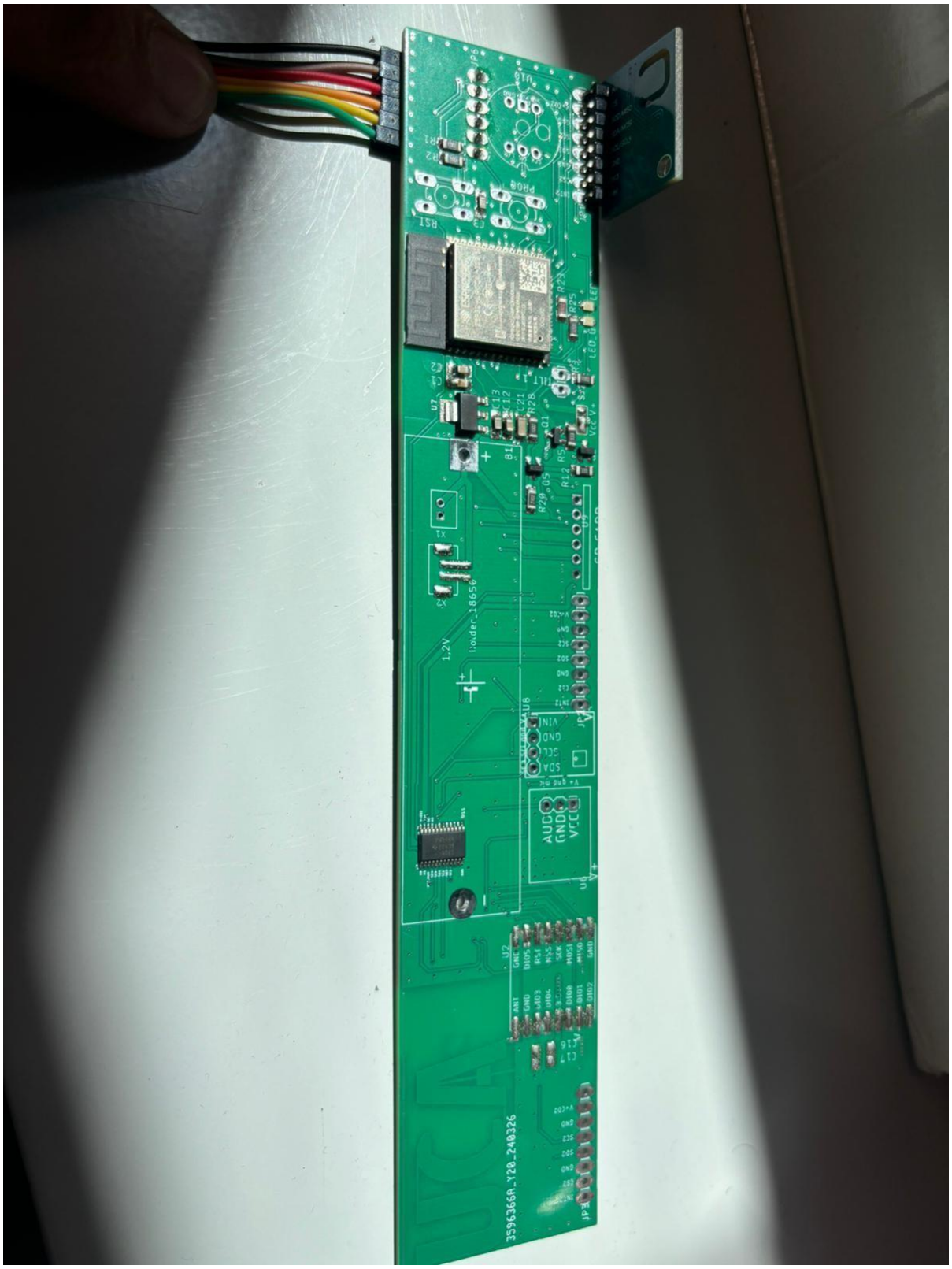
Après réception de la carte du fabricant, nous avons entrepris la soudure des composants, ainsi que la connexion d'un seul capteur pour effectuer des tests. Initialement, la carte semblait fonctionner correctement au niveau des LED et des connexions. Cependant, malheureusement, nous avons rencontré un obstacle majeur : nous n'avons pas réussi à détecter l'adresse I2C de ce capteur à l'aide de notre sketch scanner I2C.

Malgré nos efforts pour configurer correctement la pin MEASURE en niveau logique HAUT et la pin RST du MUX en niveau HAUT, ainsi que nos tentatives de toutes les combinaisons possibles avec les broches de commande A0, A1 et A2, le capteur restait introuvable.

CARNEVALI Nicolas
ELSE4



CARNEVALI Nicolas
ELSE4



CARNEVALI Nicolas
ELSE4

Travaux menés lors de l'année 2024/2025 :

Les travaux réalisés :

J'ai donc repris le projet, cette fois seul, pour une dizaine de séance. Je vais donc reprendre là où je me suis arrêté précédemment. Pour rappel, nous n'arrivions pas à détecter le capteur bien que souder à notre carte électronique. La seule adresse détectée par notre ESP32 était 0x70, mais nous ne savions pas d'où cela provenait.

J'ai donc recommencé à m'interroger sur la nature de ce dysfonctionnement. Le capteur n'est pas alimenté ? La communication I2C est corrompue ? Le code utilisé n'est pas adéquat ? L'ESP32 est défectueux ? Une soudure mal réalisée ? Le module était défectueux ?

Cela me prendra quelques moments avant de porter attention sur les soudures du capteur. Non pas à la continuité de ces derniers mais au sens précis dans lequel le capteur avait été soudé. En effet, le capteur avait été soudé à l'envers. Le ressouder correctement résoudrait donc mon problème, du moins c'est ce que je pensais …
Après avoir dessouder puis ressouder le module incluant mes capteurs j'ai réessayé de détecter ces derniers mais sans succès. J'ai donc repris mes recherches, après de nombreuses heures d'hypothèses et de tests, le temps était compté. J'ai donc décidé de faire des tests dit « unitaire », j'ai donc préparé un banc de test comportant :

- 1 ESP32
- 1 module de capteurs

Ainsi, j'éliminerai un bon nombre de potentiel problème, une mauvaise liaison sur la carte électronique, la communication I2C, un défaut du capteur soudé sur la carte mais surtout cela me donnera un réel retour sur l'utilité de mon code informatique permettant de détecter les adresses I2C.

J'ai donc interfacé un module avec un ESP32 directement sur une breadboard. J'ai adapté mon code en fonction puis j'ai compilé/chargé mon code. Le résultat était là, j'arrivai à détecter les adresses I2C de mes capteurs. Je pouvais ainsi communiquer avec eux et retirer des données. (ANNEXE 1)

À la suite de mes tests, j'étais assuré que les modules fonctionnaient bien comme je l'avais compris mais aussi que le code utilisé était fonctionnel. Le problème venait donc de la carte électronique.

J'ai réalisé des tests de continuité sur ma carte électronique après plusieurs séances de recherches déjà menées. J'ai essayé et ressayé d'injecter mon code de détection mais toujours le même résultat, savoir : 0x70. Je me suis donc penché sur l'utilisation du multiplexeur PCA, ce dernier possède 3 entrées A0, A1 et A2, je m'étais donc basé sur le principe classique d'un multiplexeur où les bits d'entrées décide de la sortie sélectionnée. Je n'avais pas émis de doute sur cela sachant que c'est le fonctionnement normal d'un MUX.
Lors de mes tests en faisant varier les bits A0, A1 et A2, j'ai pu créer ce tableau en fonction de la sortie affichée par mon ESP32.

A0	A1	A2	Résultat
0	0	0	0x70
0	0	1	0x74
0	1	0	0x72
0	1	1	0x76
1	0	0	0x71
1	0	1	0x75
1	1	0	0x73
1	1	1	0x77

De part l'analyse de mon tableau, quelques réflexions et la lecture de la doc du multiplexeur, j'ai réalisé que ces 3 bits d'entrées servaient à modifier l'adresse I2C du multiplexeur et non pas à le commander. Il existe donc une bibliothèque Arduino afin de piloter ce multiplexeur très simplement via la communication I2C. Une fonction pour lire le port et une pour écrire le port voulu. L'adresse 0x70 était donc l'adresse du multiplexeur, je n'avais ni des problèmes sur la communication I2C ni sur la partie hardware …

A partir de ce moment, j'ai réussi à détecter mes capteurs en rajoutant la gestion du MUX dans mon code informatique (ANNEXE 2). J'ai donc pu réaliser des séries de mesures de CO_2, de température et de qualité de l'air sur 1 capteur ! (ANNEXE 3)

Cela représente un pas de géant dans le projet sachant que cette étape permettait de valider complètement :

- Le fonctionnement de la carte
- Le fonctionnement des modules de capteurs
- Le fonctionnement de notre aiguillage I2C
- Le code de test

Le prochain objectif est donc d'ajouter les 2 autres modules. Malheureusement, j'ai fait face à d'autres problèmes lors de l'ajout de ces derniers. J'avais prévu d'ajouter 1 capteur puis un autre, me laissant le temps de tester l'aiguillage réalisé par le multiplexeur avant d'apporter les 3 capteurs. Cela réduira les erreurs s'il devait y en avoir. J'ai donc soudé puis testé, aucun problème visible à déclarer, j'ai adapté le code afin de faire varier le port du multiplexeur pour qu'il puisse lire les données d'un capteur puis l'autre (ANNEXE 4).

Constatant que mes tests n'était pas défaillant, le 3ème capteur est ajouté. Lors de mes tests avec 3 capteurs, j'ai constaté une erreur dans la lecture des données du 2ème et du 3ème capteurs. Aucune valeur de l'ENS160 n'était lue, ni même actualisée. Dans le code que j'ai réalisé, on lit une donnée seulement si elle a évolué par rapport à sa dernière donnée, sinon nous n'affichons rien. Puisqu'aucunes données n'étaient lues alors aucune actualisation n'était possible donc aucun affichage ni sur le capteur 2 ni sur le 3.

Face à ce problème, ma réaction devait être des plus vives car la fin des séances approchait à grand pas. J'ai donc décidé de dessouder le dernier capteur installé et de tester, de nouveau, mes mesures à 2 capteurs. Coup de théâtre, les mesures du 2ème capteur ne fonctionnent pas !

De part le temps restant et les résultats de potentiel test incertain, nous, mon encadrant et moi-même, avons décidé de réaliser les mesures seulement sur un capteur et de mettre notre attention ailleurs en espérant trouver du temps pour pouvoir résoudre ce problème ultérieurement. Cela nous permettrait donc d'avoir une carte fonctionnelle à un capteur et une campagne de mesure plutôt que rien ... (ANNEXE5)

Le formatage des données était une étape importante du projet, afin de pouvoir récupérer ces dernières facilement et de les transformer en format csv permettant de les ranger dans un tableau assez simplement.

CARNEVALI Nicolas
ELSE4

Les mesures :

Une fois la décision prise, nous travaillerons uniquement sur un capteur, nous avons lancer une campagne de mesure. Nous avons utilisé une boîte de Pringle afin de simuler la ruche, le but initial étant de mesurer le CO2 à divers niveaux.

Avant de commencer les tests, nous avons essayé de réaliser une simulation de la ruche la plus exacte possible. Pour ce faire, nous avons créé des trous en bas de cette dernière (ANEXE 6). Puis, nous avons placé la carte verticalement dans la boîte (ANNEXE 7). Une fois installé, nous avons recouvert le haut d'un plastique hermétique (ANNEXE 8). Notre système était prêt à être tester.

Pour ce faire nous avons décidé de faire 4 types de test :

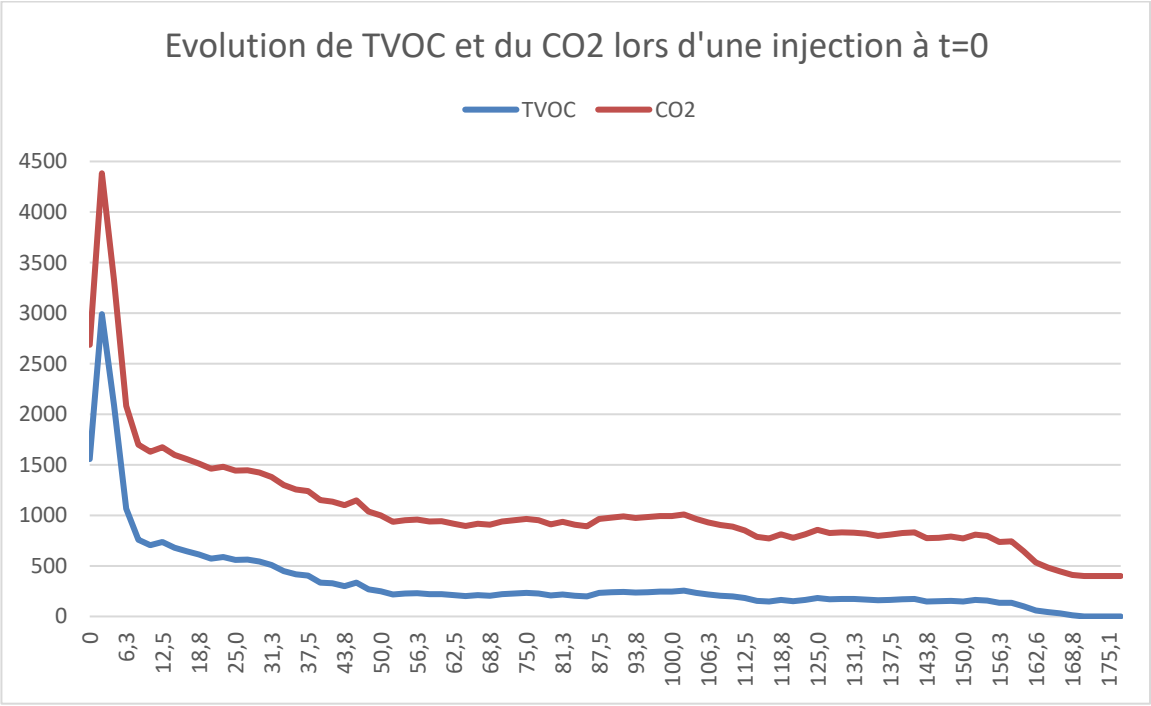
- Couvercle haut ouvert et les trous du bas bouchés avec injection du CO_2
- Couvercle haut ouvert et les trous du bas partiellement bouchés
- Couvercle haut fermé et les trous du bas partiellement bouchés puis débouché
- Couvercle haut fermé et les trous du bas non bouchés.

Nous avons donc pu retirer des données en format .csv que l'on a transformé en tableau pour chaque test.

Test numéro 1 :

Conditions : Couvercle haut ouvert et trous bouchés

Résultats :



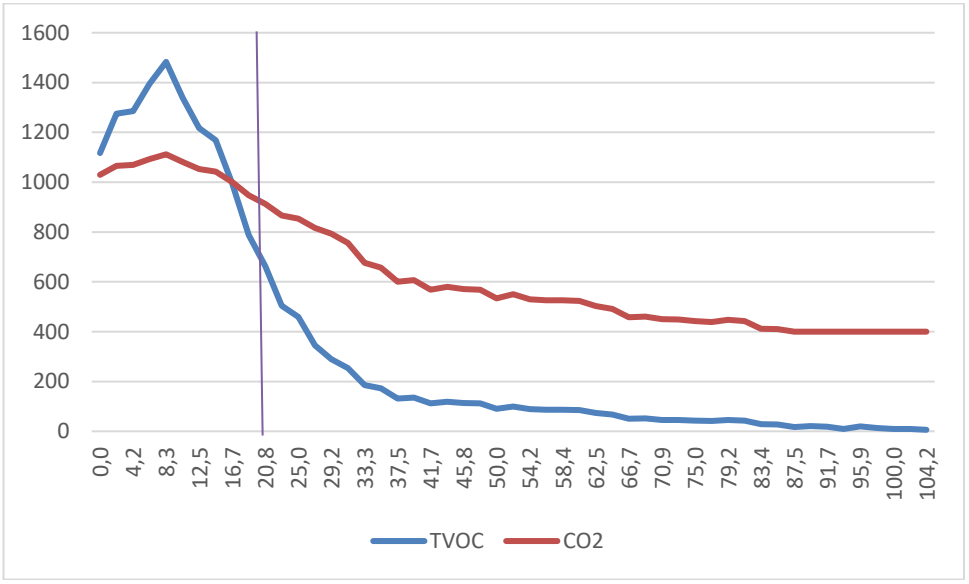
Voici une courbe représentative de notre test, on voit bien que le CO2 diminue de manière brutale lors de la fin de l'injection. Cela peut être dû à l'ouverture créée lorsque l'on retire l'appareil à injection. Sinon le CO2 diminue d'une manière plutôt constante.

CARNEVALI Nicolas
ELSE4

Test numéro 2 :

Conditions : Couvercle haut ouvert et trous bouchés

Résultats :

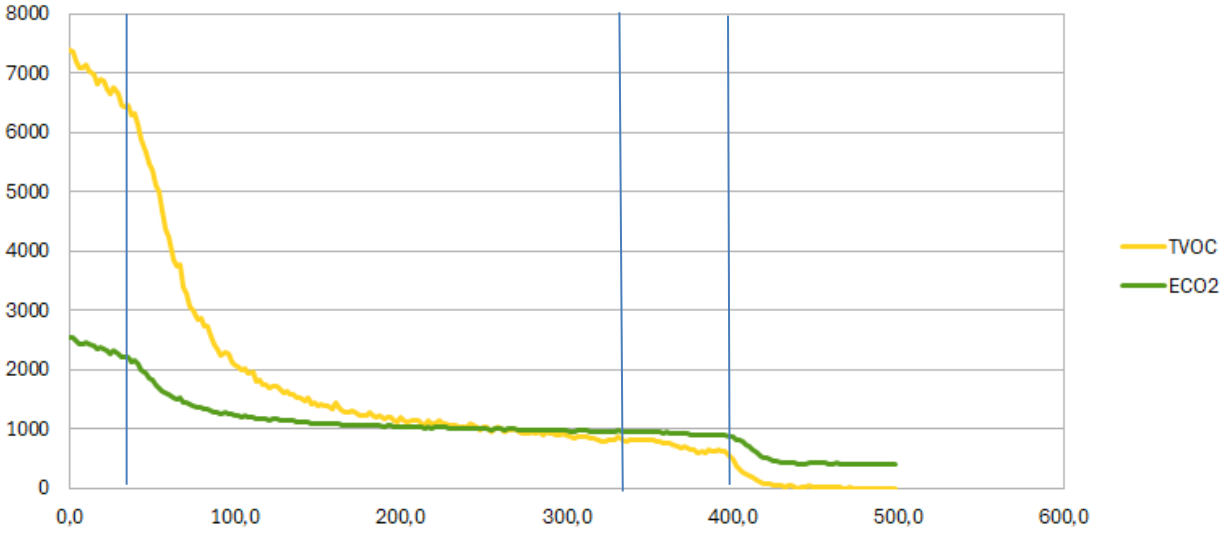


Nous pouvons constater que lorsque l'on injecte le CO2, le pic est bien moins important. Cela peu être dû à une injection moins longue ou moins importante. Nous n'allons pas tirer de conclusion sur cette donnée. Ce qui est intéressant en revanche c'est qu'à 20secondes, nous avons débouché partiellement les trous fait préalablement en bas de la boîte. Nous pouvons constater que le CO2 subit une légère décroissance supplémentaire mais rien de significatif.

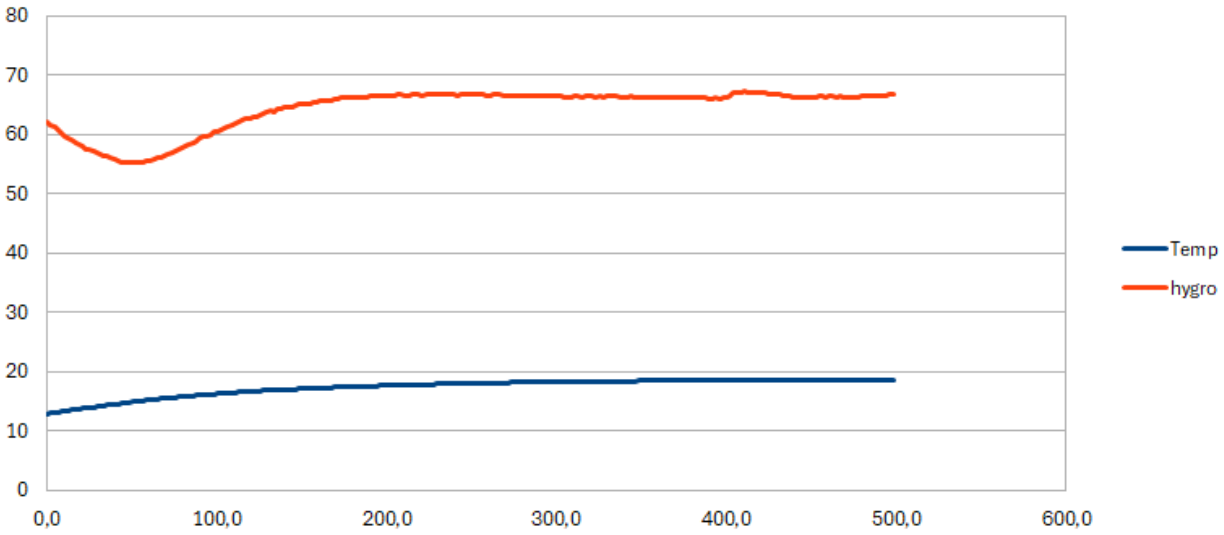
Test numéro 3 :

Conditions : Couvercle haut fermé et les trous du bas partiellement bouchés puis débouché puis ouverture couvercle.

Résultats :



CARNEVALI Nicolas
ELSE4



Ici, nous avons retiré les 4 données, les données de la température et de l'hygrométrie n'étaient pas exploitables sur les 2 premiers tests.

Sur le graphique représentant le TVOC et le CO2, on peut clairement voir 3 phases distinctes. Les lignes représentées sont de gauche à droite : ouverture partielle des trous, ouverture totale des trous, ouverture du couvercle. Nous pouvons ainsi remarquer que le fait de laisser une ouverture dans notre système est critique. Une majorité de CO2 est dissipé dans la première phase avant même d'ouvrir complètement les trous se situant en bas de la boite. Lors de l'ouverture complète de cette dernière (ouverture du couvercle), nous remarquons que le reste du CO_2 s'échappe pour tomber à la normale dans l'air (400).
Cela rendra notre régulation de CO2 assez complexe à maintenir.

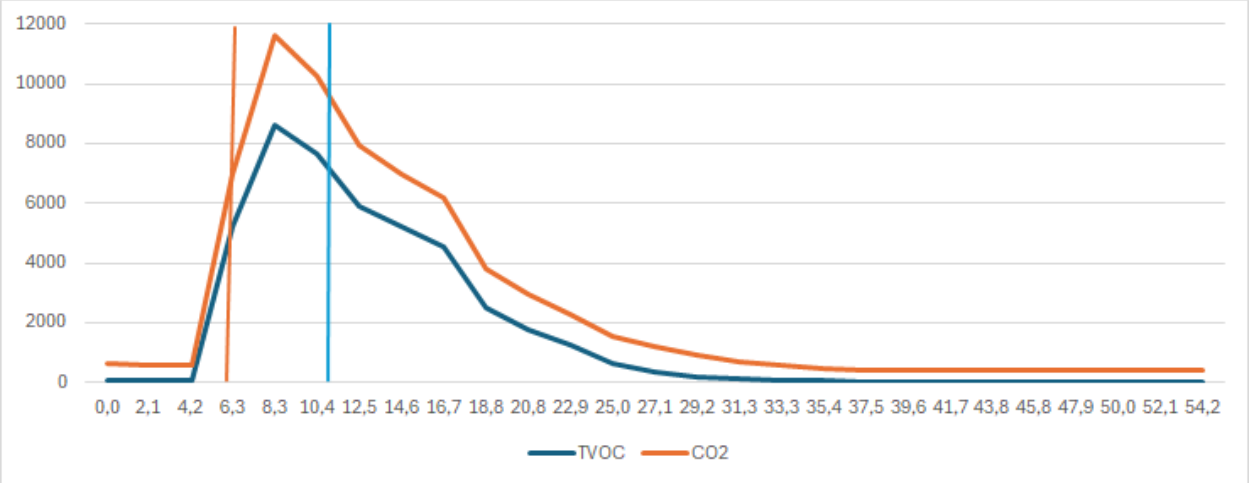
Passons à l'analyse de l'hygrométrie et de la température. Nous constatons que l'hygrométrie varie légèrement au début, lorsque la boite est complètement hermétique puis retrouve une valeur dites « classique » lorsque l'on ouvre partiellement les trous situés en bas de la boite. Il faudra donc faire attention à ne pas laisser la ruche fermée hermétiquement trop longtemps lorsque nous injecterons le CO2 afin de ne pas atteindre des seuils d'hygrométrie trop bas. Nous pouvons constater également que la température à quasiment doublé durant l'expérience. Elle a varié de 6°C en seulement 500 secondes soit moins d'une dizaine de minutes. Cela pourra être un bon point comme un mauvais pour les abeilles car pour faire augmenter la température de la ruche, ces dernières créent du mouvement. Avoir une température trop souvent élevée pourrait résulter à une inactivité des abeilles dans la ruche.

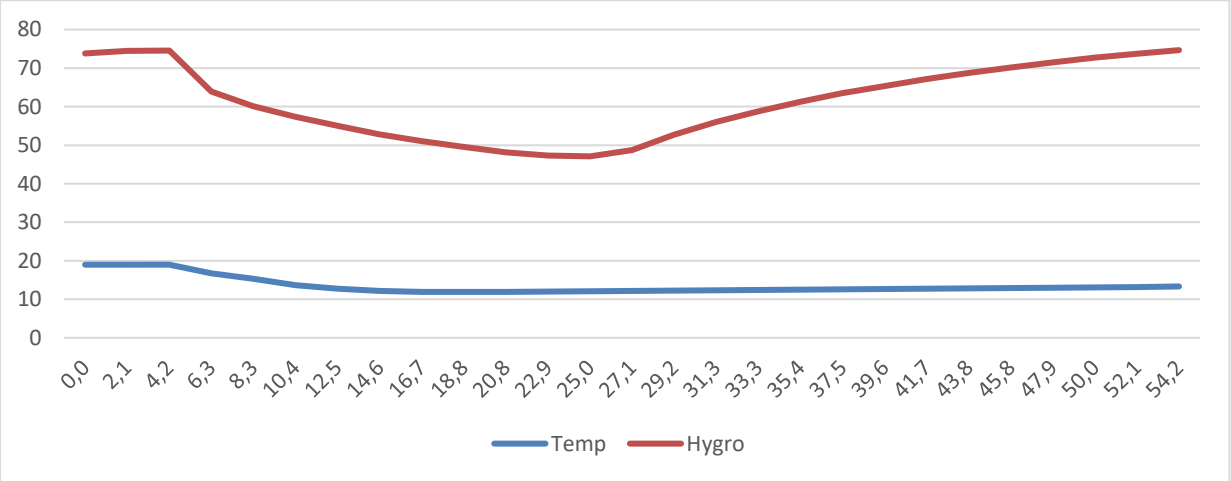
CARNEVALI Nicolas
ELSE4

Test numéro 4 :

Conditions : Couvercle haut fermé et les trous du bas non bouchés.

Résultats :





Ici, l'injection commence environ à 6 secondes par des jets successifs, elle se termine à environ 10s. Comme les trous sont complètement ouverts nous pouvons voir une dissipation du CO_2 assez importante et rapide. Le taux de CO_2 est redevenu normal moins d'une minute après ce qui n'est pas étonnant sachant que plus de la moitié du CO_2 s'est dissipé en approximativement 5min lors du test précédent alors que 5x moins de trous était alors ouverts.
L'hygrométrie diminue lorsque le CO_2 est injecté puis croit progressivement car les trous sont complètement ouverts.

Ce qui est étonnant, en revanche c'est le comportement de la température. Nous avions vu précédemment que cette dernière augmentait en fonction du taux de CO_2 présent. Or, ici, la température diminue puis reste constante.
Cela peut être dû au fait d'avoir enchainer les tests et donc peut être une température dans la boite plutôt élevée au début du test.

CARNEVALI Nicolas
ELSE4

Conclusion :

La menace que représente les Varroas pour les colonies d'abeilles demeure un enjeu préoccupant aux répercussions écologiques et économiques considérables. Face à l'inefficacité croissante des traitements conventionnels, souvent limités par des phénomènes de résistance ou par leur toxicité pour les abeilles, il est impératif d'explorer des solutions novatrices et respectueuses de l'équilibre apicole. Dans cette optique, notre étude s'est attachée à évaluer le potentiel d'une méthode de lutte alternative, fondée sur l'injection contrôlée de dioxyde de carbone (CO_2), en exploitant la différence de sensibilité entre le varroa et son hôte.

Les résultats obtenus confirment que le CO_2 peut constituer un levier sélectif prometteur pour affaiblir le parasite sans compromettre significativement la santé des abeilles. Cette approche offre ainsi une piste intéressante vers un mode de gestion plus durable des infestations, tout en minimisant les perturbations au sein de la ruche. Toutefois, des recherches complémentaires sont nécessaires pour affiner les protocoles d'application, évaluer les effets à long terme et garantir l'intégration harmonieuse de cette méthode dans les pratiques apicoles existantes.

En conclusion, ce travail ouvre des perspectives encourageantes pour le développement de stratégies de lutte ciblée contre le varroa, conciliant efficacité, innocuité et durabilité, au service d'une apiculture plus résiliente et respectueuse des écosystèmes.

CARNEVALI Nicolas
ELSE4

ANNEXES :

ANNEXE 1 :

```
 8  void setup() {
 9  Serial.begin(115200);
10  Serial.println("\nI2C Scanner");
11  pinMode(ADD, OUTPUT);
12  pinMode(CS, OUTPUT);
13  digitalWrite(ADD,LOW);
14  digitalWrite(CS, HIGH);
15  delay(1000);
16  Wire.begin(SDA, SCL);
17  }
18  void loop() {
19    byte error, address;
20    int nDevices;
21    Serial.println("Scanning...");
22    nDevices = 0;
23    for(address = 1; address < 127; address++ ) {
24      Wire.beginTransmission(address);
25      error = Wire.endTransmission();
26      if (error == 0) {
27        Serial.print("I2C device found at address 0x");
28        if (address<16) {
29          Serial.print("0");
30        }
31        Serial.println(address,HEX);
32        nDevices++;
33      }
34      else if (error==4) {
35        Serial.print("Unknow error at address 0x");
36        if (address<16) {
37          Serial.print("0");
38        }
39        Serial.println(address,HEX);
40      }
41    }
42    if (nDevices == 0) {
43      Serial.println("No I2C devices found\n");
44    }
45    else {
46      Serial.println("done\n");
47    }
48    delay(5000);
49  }
50
```

CARNEVALI Nicolas
ELSE4

ANNEXE 2 :

```ino
#include <Wire.h>
#include <SparkFun_I2C_Mux_Arduino_Library.h>


QWIICMUX myMux;

#define SCL 22
#define SDA 21
#define CS0 16
#define CS1 26
#define CS2 27
#define A0 2
#define A1 4
#define A2 12
#define RST 15
#define MEAS 13
#define I2C_ADDRESS 0x52

void setup() {
Serial.begin(115200);
Serial.println("\nI2C Scanner");
pinMode(RST,OUTPUT);
pinMode(MEAS,OUTPUT);
pinMode(CS0, OUTPUT);
pinMode(CS1, OUTPUT);
pinMode(CS2, OUTPUT);
pinMode(A0, OUTPUT);
pinMode(A1, OUTPUT);
pinMode(A2, OUTPUT);
digitalWrite(RST, HIGH);
digitalWrite(MEAS,HIGH);
digitalWrite(CS0, HIGH);
digitalWrite(CS1, HIGH);
digitalWrite(CS2, HIGH);
digitalWrite(A0,LOW);
digitalWrite(A1,LOW);
digitalWrite(A2,LOW);
delay(1000);
Wire.begin(SDA, SCL);

if (myMux.begin(0x70,Wire) == false)
  {
    Serial.println("Mux not detected. Freezing...");
    Serial.println("Mux not detected. Freezing...");
    while (1)
      ;
  }
Serial.println("Mux detected");
myMux.setPort(0);
}

void loop() {
  byte error, address;
  int nDevices;
  Serial.println("Scanning...");
  nDevices = 0;

  for(address = 1; address < 127; address++ ) {
     //Serial.println("1");
    Wire.beginTransmission(address);
    //Serial.println("2");
    error = Wire.endTransmission();
    //Serial.println("3");
    if (error == 0) {
      Serial.print("I2C device found at address 0x");
      if (address<16) {
        Serial.print("0");
      }
      Serial.println(address,HEX);
      nDevices++;
    }
    else if (error==4) {
      Serial.print("Unknow error at address 0x");
      if (address<16) {
        Serial.print("0");
      }
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0) {
    Serial.println("No I2C devices found\n");
  }
  else {
    Serial.println("done\n");
  }
  delay(5000);
}
```

CARNEVALI Nicolas
ELSE4

ANNEXE 3 :

```ino
#include <Wire.h>
#include <SparkFun_I2C_Mux_Arduino_Library.h>
#include <Arduino.h>
#include <ScioSense_ENS16x.h>
#include <AHT20.h>

QWIICMUX myMux;
ENS160 ens160;
AHT20 aht20;

#define SCL 22
#define SDA 21
#define CS0 16
#define CS1 26
#define CS2 27
#define A0 2
#define A1 4
#define A2 12
#define RST 15
#define MEAS 13
#define I2C_ADDRESS_ENS 0x52
#define I2C_ADDRESS_AHT 0x38

void setup() {
Serial.begin(115200);
pinMode(RST,OUTPUT);
pinMode(MEAS,OUTPUT);
pinMode(CS0, OUTPUT);
pinMode(CS1, OUTPUT);
pinMode(CS2, OUTPUT);
pinMode(A0, OUTPUT);
pinMode(A1, OUTPUT);
pinMode(A2, OUTPUT);
digitalWrite(RST, HIGH);
digitalWrite(MEAS,HIGH);
digitalWrite(CS0, HIGH);
digitalWrite(CS1, HIGH);
digitalWrite(CS2, HIGH);
digitalWrite(A0,LOW);
digitalWrite(A1,LOW);
digitalWrite(A2,LOW);
delay(1000);
Wire.begin(SDA, SCL);
Serial.println("\nDétection Multiplexeur ..");

if (myMux.begin(0x70,Wire) == false)
if (myMux.begin(0x70,Wire) == false)
  {
    Serial.println("Mux not detected. Freezing...");
    while (1)
      ;
  }
Serial.println("Mux detected");
myMux.setPort(0);
ens160.begin(&Wire, I2C_ADDRESS_ENS);
Serial.println("begin..");
while (ens160.init() != true)
    {
        Serial.print(".");
        delay(1000);
    }

Serial.println("ENS connection successful");
ens160.startStandardMeasure();
if (aht20.begin() == false)
  {
    Serial.println("AHT20 not detected. Please check wiring. Freezing.");
    Serial.print(".");
        delay(1000);
  }
Serial.println("AHT connection successful");
}

void loop() {
  // put your main code here, to run repeatedly:
  ens160.wait();
  Serial.print("\nDébut des mesures\n");
  if (ens160.update() == RESULT_OK)
     {
        if (ens160.hasNewData())
        {
            Serial.print("AQI UBA:"); Serial.print((uint8_t)ens160.getAirQualityIndex_UBA());
            Serial.print(" TVOC:"); Serial.print(ens160.getTvoc());
            Serial.print(" ECO2:"); Serial.println(ens160.getEco2());
        }
     }
  Serial.print("\nT: "); Serial.print(aht20.getTemperature(),2);
  Serial.print("°C\nH: "); Serial.print(aht20.getHumidity(),2);
  Serial.print("%\n");
  delay(1000);
}
```

CARNEVALI Nicolas
ELSE4

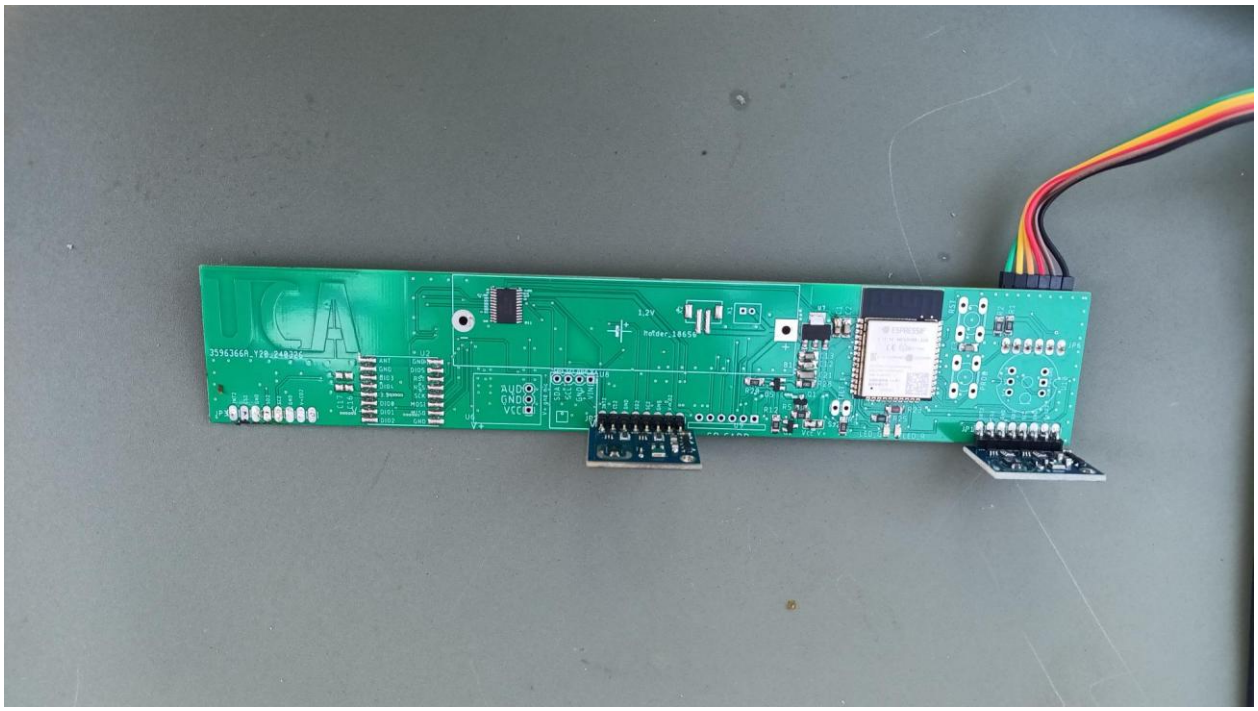
ANNEXE 4 :

```
byte error, address;
int nDevices;
byte MyCurrentPort;

MyCurrentPort = myMux.getPort();
Serial.print("Current Port of MUX : ");Serial.println(MyCurrentPort);
if(MyCurrentPort == 0)
{
  digitalWrite(LED_R,LOW);
  digitalWrite(LED_G,LOW);
}
else if (MyCurrentPort == 1)
{
  digitalWrite(LED_R,HIGH);
  digitalWrite(LED_G,LOW);
}
else if (MyCurrentPort ==2){
  digitalWrite(LED_R,HIGH);
  digitalWrite(LED_G,HIGH);
}
```

```
if (MyCurrentPort == 0)
  {
    myMux.setPort(1);
  }
  else if (MyCurrentPort == 1)
  {
    myMux.setPort(2);
  }
  else if (MyCurrentPort == 2)
  {
    myMux.setPort(0);
  }
```

ANNEXE 5 :



CARNEVALI Nicolas
ELSE4

ANNEXE 6 :

ANNEXE 7 :

CARNEVALI Nicolas
ELSE4

ANNEXE 8 :