Dossier technique

Conception d’un banc de test de batteries

Fait le 20/01/2025

Table of Contents

Introduction

Ce document contient l'ensemble des informations techniques relatives au projet de conception d'un banc de cyclage de batterie fait par deux étudiants de l'école POLYTECH. Il doit permettre entre autres de trouver :
- Les documents sources du projet.
- Référencement des composants utilisés.
- Procédure de mise en route.
- Schéma ; PCB.
- Feuilles de calcul de conception et dimensionnement

Dans ce document, tous les efforts ont été mis en œuvre pour éviter les anomalies. Toutefois, nous ne pouvons garantir que ce document soit à 100% exempt de toute erreur. Les informations présentes dans ce dossier sont strictement données à titre indicatif.

1. Hardware

1.1. Architecture interne du banc de cyclage

La stratégie pour acquérir les données de tension, courant et température est d'utiliser un module de contrôle et d'acquisition de données distribué et développé par Advantech, conçu spécifiquement pour les applications industrielles.
La carte électronique **ADAM-5000**, pour surveiller, contrôler et collecter des données provenant de divers équipements et capteurs.

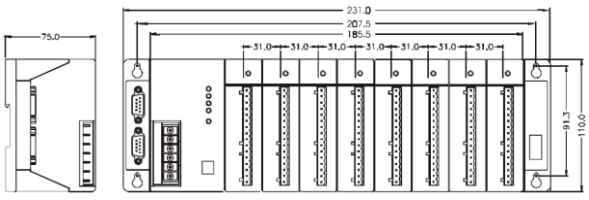


Figure 2. Carte ADAM-5000

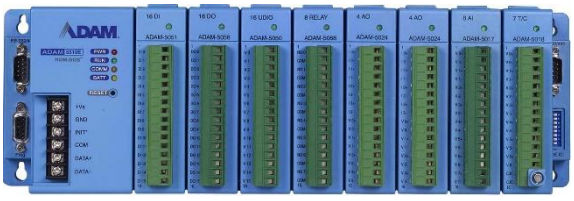


Figure 1. Carte ADAM-5000 illustration

Cette carte recevra 4 modules :

- 1xModule ADAM-5017 : 8 entrées analogiques différentielles (+/- 10 V) pour les mesures de courants et tensions.
- 2xModule ADAM-5018 : 2x7 entrées thermocouples pour la mesure de température.

- 1xModule ADAM-5060 : 6 sorties numériques pouvant délivrer jusqu'à 30 V et 2 A pour le pilotage du relais de sécurité.

Ces cartes ADAM sont adaptées pour un usage industriel et dispose d'une robustesse ; protection surtension de 30 V et isolation galvanique de 3000 V entre les entrées/sorties, de plus, elle fonctionne par protocole Ethernet, pratique élaborer un réseau local et interconnecter les composants.

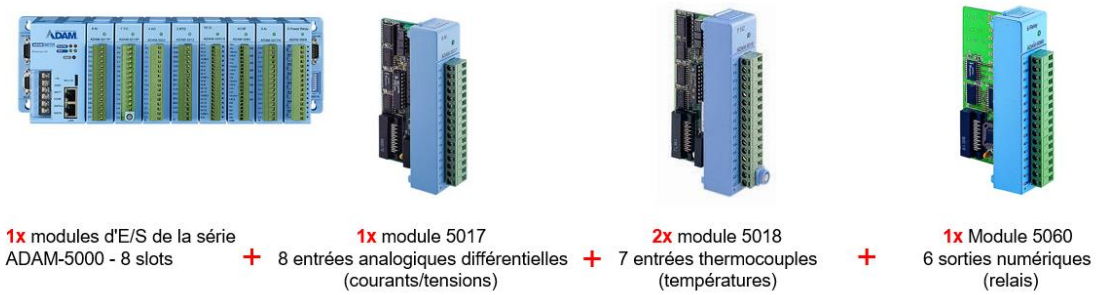


Figure 3. Modules ADAM-5000

1.2. Capteurs

Vous trouverez les capteurs choisis qui sont compatibles avec ces cartes d'acquisition et qui respectent le cahier des charges, tout en prenant le soin de prendre en compte le critère économique :

La mesure du courant dans une plage de +/- 800 A se fera avec les capteurs transducteurs pour courant continue et alternatif LEM HAL-300S, capable de mesurer +/- 800 A avec une précision de 1%, (par rapport au courant nominal 300 A) soit +/- 0,3 A.
Ils s'alimentent en +/- 15 V, ils fournissent en sortie une tension de +/- 4 V en rapport de proportionnalité au courant mesuré. Il sera possible de brancher ces capteurs dans les entrées différentielles analogiques de la carte ADAM-5017 et convertir numériquement la valeur de la tension de sortie du capteur pour interpréter la valeur du courant de l'équipement sous test mesurée.

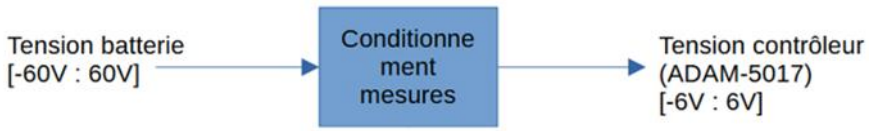
Figure 4. Capteur de courant

La mesure de la température se réalisera à l'aide de 10 thermocouples à jonction exposée de marque RS PRO à raccord dénudé, type T pour la compatibilité de traitement avec le module ADAM-5018. La plage de mesure est de -75 °C à +250 °C (correspond au cahier des charges).

Figure 5. Thermocouples

La mesure de tension se réalisera à l'aide d'une carte électronique d'instrumentation développée par les étudiants, qui réalisera les fonctions suivantes :

- Adaptation de la tension de 60 V à 6 V.
- Filtrage des perturbations en entrée et en sortie de la carte.
- Isolation galvanique entre la batterie (entrée) et la sortie.



Pour imprimer et fabriquer cette carte, le schéma électrique est fourni en annexe (Annexe II), il suffira de reprendre le schéma et les composants référencés en annexe (Annexe III) multiplié par trois et obtenir les 3 canaux requis.

Tous les composants choisis sont en traversants pour la facilité de soudure et l'expérimentation sur plaque. Les microcontrôleurs sont capables de lire des tensions analogiques, ainsi pour mesurer la tension de la batterie, il suffira d'adapter le niveau de tension de la batterie (+/- 60 V) au niveau de tension admissible par le microcontrôleur (+/- 10 V), par simplicité, nous choisissons un facteur de division par 10 (+/- 6 V).

Le première étage consiste à insérer un pont diviseur de tension résistif pour diviser la tension par 10, de plus, on insère une capacité de filtrage de 200 nF en parallèle de la résistance de 90 kOhms, on obtient ainsi, un filtre passe-bas du premier ordre avec une fréquence de coupure de 9 Hz. Ce filtre du premier ordre est capable de filtrer tout type de perturbations indésirables qui viendrait s'ajouter à la mesure, sachant que la

fréquence d'acquisition imposée est de 1 Hz, le choix de 9 Hz est judicieux pour conserver seulement la grandeur continue de la tension.

Le troisième étage consiste à insérer un amplificateur soustracteur avec un gain de 1 qui va réaliser la mesure différentielle entre les deux potentiels. Ce composant se présente sous la forme d'un circuit intégré Amplificateur d'instrumentation contenant à l'intérieur 3 amplificateurs pour la stabilité et la fiabilité, pour choisir un gain de 1, la datasheet indique de laisser une impédance infinie (pas de résistance, pin à vide) entre les broches R_G+ et R_G- :

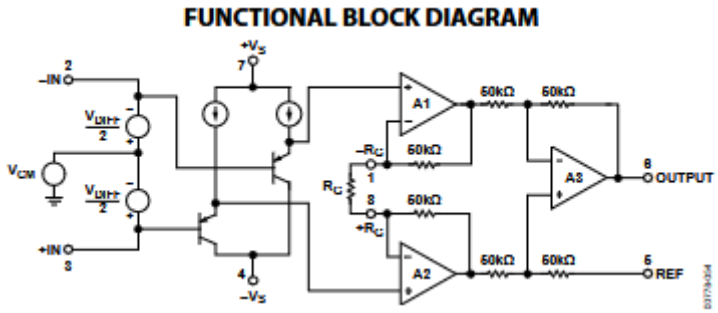


Figure 6. Schéma fonctionnel provenant de la datasheet AD623

Datasheet AD623 : https://www.analog.com/media/en/technical-documentation/data-sheets/ad623.pdf

Le quatrième étage consiste à ajouter un amplificateur d'isolement de 1500 V entre la mesure de la batterie et la sortie de la carte avec un gain de 1.

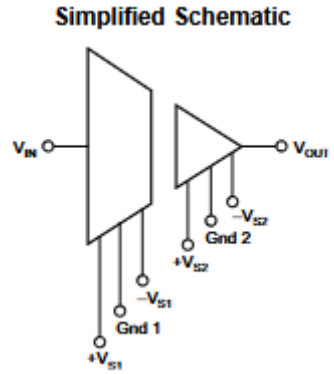


Figure 7. Schéma simplifié de la datasheet AOP ISO124P

Il y aura également un régulateur de tension isolé pour l'isolation galvanique complète : Datasheet TMH 2415D : https://docs.rs-online.com/67cb/0900766b8172f1f9.pdf

L'impédance du module ADAM-5017 qui sera connecté en sortie de cette carte électronique d'instrumentation a une impédance d'entrée de 1 MOhms, on peut donc ajouter un autre filtre passe-bas avec une résistance de 1 kOhms et une capacité de 200 nF pour filtrer les perturbations, pour filtrer les perturbations qui viendrait s'ajouter sur les câbles de la sortie (fréquence de coupure de 800 Hz, suffisante pour filtrer les perturbations qui se situent à 2 kHz).

1.4. Alimentation

Pour la partie basse-puissance, il est nécessaire d'alimenter :

- Les capteurs de courant en +/- 15 V
- Les AOP de la carte d'instrumentation avec la même alimentation
- Les modules ADAM-5000 en 12 V
- Le BMS de la batterie en 12 V

Nous choisirons donc une alimentation avec les sorties : + 15 V, - 15 V et + 12 V.

Pour l'alimentation 12 V, nous choisissons une alimentation de 80 W pour le BMS et tous les modules ADAM largement suffisante, qui doit se brancher en entrée sur le réseau par raccords dénudés sur bornier et en sortie également :

Figure 8. Alimentation à découpage 12 V, 80W

Pour l'alimentation +/- 15 V, nous choisissons une alimentation TRACOPOWER de 65 W largement suffisante pour alimenter les capteurs de courant et les AOP de la carte de mesure de tension :

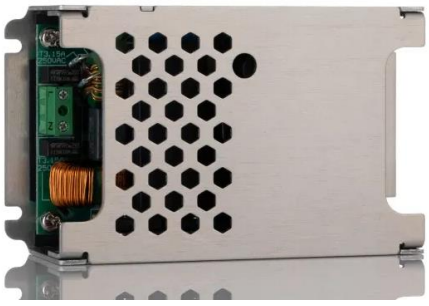


Figure 9. Alimentation à découpage +/- 15 V, 65W

L'entrée sera le réseau monophasé par raccord dénudé sur bornier et la sortie sera +/- 15 V par bornier également.

1.5. Sécurité

On pourra couper l'alimentation de l'alimentation bidirectionnelle pilotant les charges et les décharges de batterie ;

- On utilisera un commutateur électromécanique triphasé piloté par tension de bobine connecté au réseau en entrée et à l'alimentation bidirectionnelle en sortie.
- Pour commander la commutation, la bobine du contacteur sera connectée au réseau en série avec un bouton poussoir normalement fermé, lors de l'appui sur le bouton, la position ouverte se maintiendra et la bobine du contacteur n'étant plus alimenté, ouvrira le circuit triphasé.
- Pour communiquer de l'appui de l'arrêt d'urgence, un régulateur 230 Vac vers 5 Vdc ne sera lui non-plus plus alimenté, ce qui indiquera un « 0 » logique pour la carte d'acquisition module ADAM-5017. La programmation de ce module devra prévoir une de ces entrées pour relier ce signal de 5 V et déclencher un évènement lorsqu'elle détecte 0 V. Afin d'avertir l'utilisateur et arrêter l'acquisition de donnée.

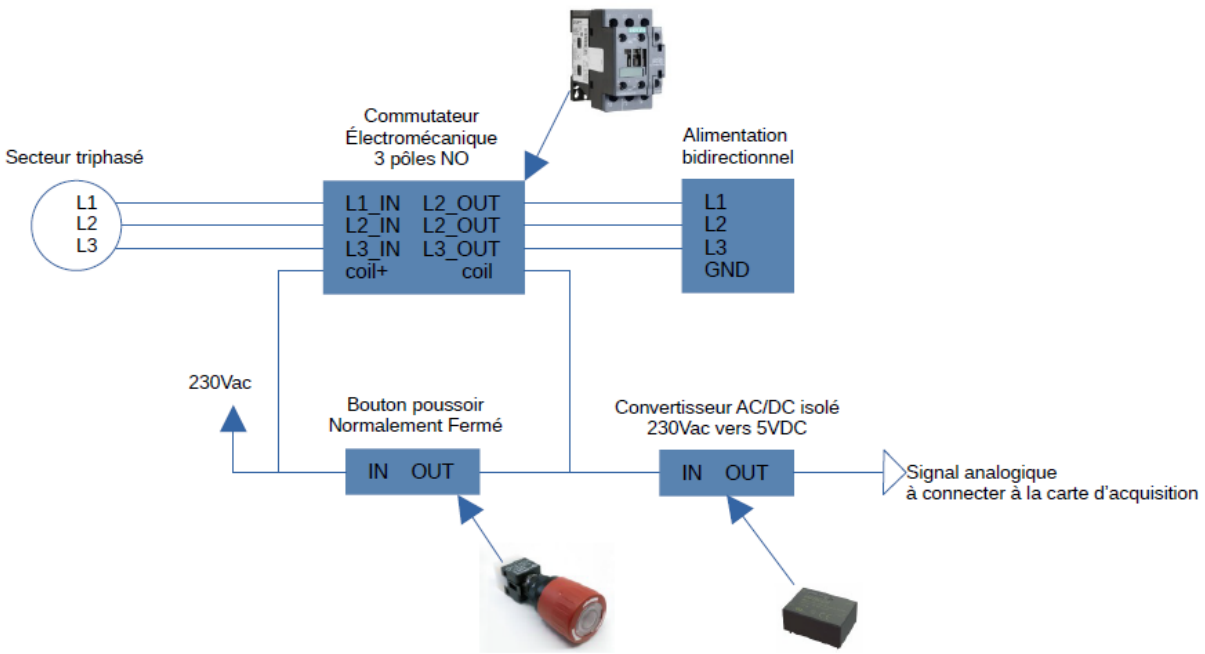


Figure 10. Schéma fonctionnel Arrêt urgence matériel

On pourra également couper le circuit de connexion de la batterie par un arrêt d'urgence logiciel, piloté par le module ADAM-5060, qui est capable de fournir jusqu'à 2 A et 30 V en sortie.

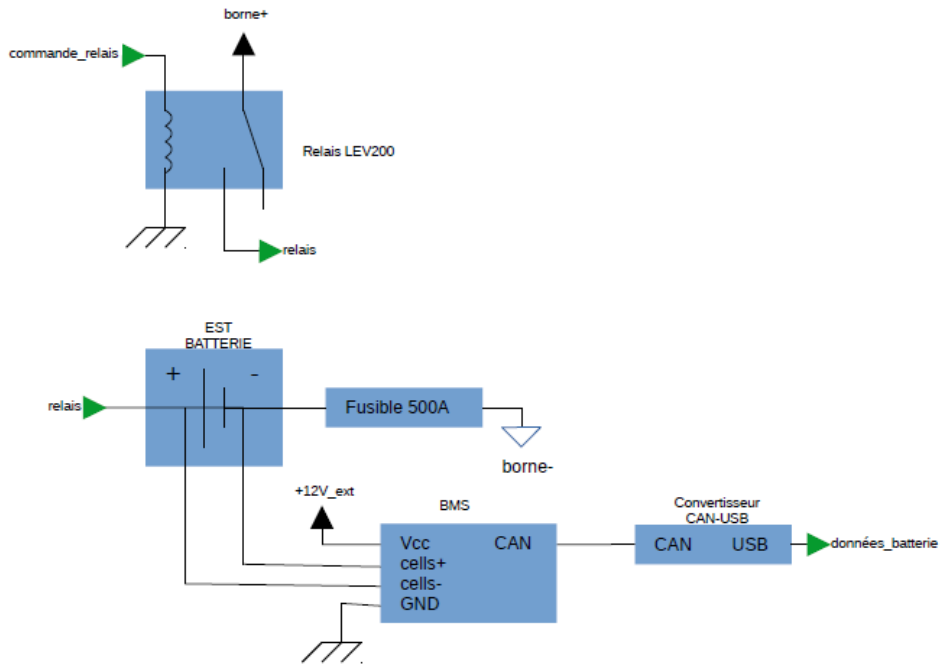


Figure 11. Schéma fonctionnel arrêt d'urgence logiciel

Le relais pour positionner la batterie à vide ou en charge/décharge, sera piloté à l'aide d'une tension de bobinage de 12 V et un courant minimal de 100 mA selon sa documentation :
http://www.thunderstruck-ev.com/Manuals/LEV200%20Series%20Data%20Sheet.pdf
Le module de sortie numérique ADAM-5060 peut fournir jusqu'à 2 A selon sa documentation :
https://advdownload.advantech.com/productfile/PIS/ADAM-5069/Product%20-%20Datasheet/ADAM-5069_DS20140102161355.pdf
Il n'y aura donc pas besoin d'un étage de puissance entre les deux.

2. Software

2.1. Communication Ordinateur-BMS

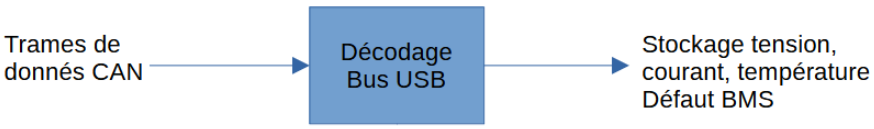
Pour tester une batterie, il faut communiquer avec son BMS (battery management system) afin de récupérer les données sensibles telles que la tension cellule, le courant, la température et les défauts BMS. Pour cela, nous devons établir une communication

entre le bus CAN du BMS et le bus USB de l'ordinateur, nous utiliserons un adaptateur type convertisseur de données CAN vers USB :

Figure 12. Adaptateur pour communiquer entre BMS et ordinateur

Pour réaliser cette fonctionnalité, il faut :

- Écrire un programme pour initialiser le périphérique ou utiliser les logiciels et drivers directement fournis par le fabricant dans la documentation : https://docs.rs-online.com/c4e2/A700000009169933.pdf
- Écrire un programme pour décoder les trames de données CAN et les lire sur l'ordinateur :



Le programme est disponible en annexe (annexe V).

2.2. Communication Ordinateur-Alimentation

Au sein d'un banc de cyclage, l'alimentation bidirectionnelle est un élément clé. En effet, celle-ci permet l'exécution des différents cycles de charge/décharge de l'EST (Équipement Sous Test), justifiant l'existence de ce genre de technologies. L'assurance d'une bonne communication avec l'alimentation et la bonne configuration des cycles sont donc deux étapes cruciales du projet.

Comme stipulé dans le cahier des charges, l'alimentation qui sera utilisée tout au long de ce processus est de la gamme Elektro-Automatik EA-PSB 10000.
D'après les indications du client, le mode de communication que l'on utilisera sera un protocole Ethernet. Pour cela, il est nécessaire de se connecter à l'alimentation en deux phases :

- De manière filaire, donc à l'aide d'un câble RJ45 branché à un port LAN (Ethernet) de l'alimentation.
- De manière numérique, aux mêmes réseaux que l'alimentation, donc en configurant sur l'ordinateur qui communique : le service réseaux (le nom du câble utilisé), le type de configuration IP (ici, « Manuellement », pour pouvoir modifier le reste des paramètres), l'adresse IP (propre à l'ordinateur), le masque de sous-réseau et le routeur (qui doivent être les mêmes que ceux de l'alimentation).

Figure 13. Paramétrage de la communication

D'après l'écran d'affichage de l'alimentation, l'adresse IP utilisée par défaut par celle-ci est « 192.168.0.3 », le routeur est « 192.168.0.1 » et le masque de sous-réseau est « 255.255.255.0 ». Il est alors nécessaire de choisir une adresse IP pour notre ordinateur de la forme « 192.168.0.X » car le masque indique que l'identifiant de l'appareil (ce qui le différencie des autres appareils sur ce réseau) est la dernière valeur de son adresse IP (seul le dernier numéro du masque est différent de 255). Ce choix permettra à l'alimentation de reconnaitre l'ordinateur.

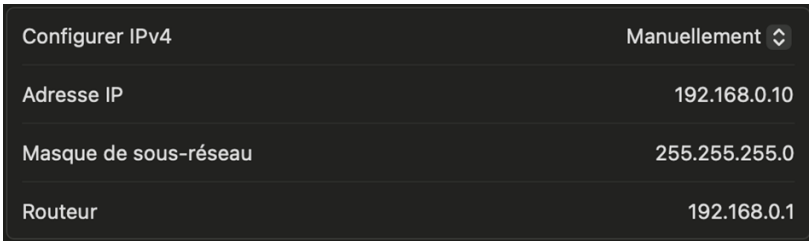


Figure 14. Paramétrage réseaux de l'ordinateur

Une fois ceci fait, il est nécessaire de choisir un protocole de communication parmi ceux existants en mode Ethernet. Afin de pouvoir coder les différentes phases de cycles en Python et de facilement envoyer les différentes commandes, on a fait le choix d'utiliser

le protocole SCPI. En effet, celui-ci permet, avec une syntaxe simple, d'envoyer les différentes instructions à l'alimentation et ainsi, de réaliser les cycles.

L'intérêt premier du choix du protocole (et donc de la syntaxe de communication), est de pouvoir assurer que l'alimentation est capable de répondre lorsqu'on lui envoie une requête (lorsqu'on lui « pose une question »). Le cas échéant, certaines commandes des anciens standards GPIB et IEEE 488 sont implémentées et prises en charge par tous les appareils utilisant le langage de commande SCPI. Celle qui nous intéresse est la commande « *IDN ? ».

Lorsqu'elle est correctement perçue par l'appareil, celui-ci doit renvoyer sa chaîne d'identification, qui contient les informations suivantes, séparées par des virgules :

1. Fabricant.
2. Nom du modèle.
3. Numéro de série.
4. Version(s) du firmware (s'il y en a plusieurs, elles sont séparées par un espace).

Pour résumer, si l'alimentation nous renvoie ces informations à la suite de l'appel de la requête « *IDN ? », nous pourrons statuer sur la communication avec l'appareil et commencer à lui envoyer des instructions.

```
Connecté à l'alimentation.
Commande: *IDN?
Réponse: EA Elektro-Automatik GmbH & Co. KG, ELR 10200-50, 2775120003, V3.03 05.10.2022 V3.06 24.05.2023 V1.0.2.20,
```

Figure 15. Vérification de dialogue

Comme imposé précédemment, l'alimentation nous répond bien avec les différentes informations attendues. La communication a donc bien été assurée. Nous pouvons passer à la commande de la sortie bidirectionnelle.

Afin d'être capable d'imposer des valeurs de sortie à distance (de charge et de décharge), il est nécessaire de faire passer l'alimentation en mode « remote », soit « télécommande » en anglais, signifiant qu'elle sera maintenant capable d'être contrôlée à distance (même si nous sommes branchés physiquement à l'appareil, nous ne faisons pas varier manuellement les valeurs de sorties analogiques). L'envoie de la commande « SYST:LOCK ON » permet le passage à ce mode.

```
Connecté à l'alimentation.
Commande: *IDN?
Réponse: EA Elektro-Automatik GmbH & Co. KG, ELR 10200-50, 2775120003, V3.03 05.10.2022 V3.06 24.05.2023 V1.0.2.2
0,
Commande: SYST:LOCK ON
Réponse:
```

Figure 16. Commande de passage en mode remote

L'inconvénient de l'utilisation d'une bibliothèque comme « socket » est qu'elle présente des délais d'expiration de la connexion pour l'interface Ethernet. Ainsi, le réglage à 0 de ce « timeout » désactive le délai d'expiration (le réglage par défaut est de 5s). Nous utilisons donc la commande « SYST : TIMEOUT 0 » qui permet ce réglage.

```
Connecté à l'alimentation.
Commande: *IDN?
Réponse: EA Elektro-Automatik GmbH & Co. KG, ELR 10200-50, 2775120003, V3.03 05.10.2022 V3.06 24.05.2023 V1.0.2.2
0,
Commande: SYST:LOCK ON
Réponse:
Commande: SYST:TIMEOUT 0
Réponse:
```

Figure 17. Commande d'arrêt des délais d'expiration de la connexion

Comme la programmation de l'alimentation s'est faite avec une suite d'erreurs commises, il est important de supprimer la file d'attente des erreurs pour ne pas surcharger la mémoire de l'alimentation. Pour cela, nous utiliserons la commande « *CLS ».
Dans le même registre, nous allons utiliser la commande « *RST » qui permet de remettre l'alimentation dans un état par défaut.
Nous pouvons maintenant activer la sortie DC de l'appareil. La commande « OUTP ON » permet de faire ceci si le contrôle à distance est actif.

```
Connecté à l'alimentation.
Commande: *IDN?
Réponse: EA Elektro-Automatik GmbH & Co. KG, ELR 10200-50, 2775120003, V3.03 05.10.2022 V3.06 24.05.2023 V1.0.2.2
0,
Commande: SYST:LOCK ON
Réponse:
Commande: SYST:TIMEOUT 0
Réponse:
Commande: *CLS
Réponse:
Commande: *RST
Réponse:
Commande: OUTP ON
Réponse:
```

Figure 18. Commande d'activation de la sortie DC

L'alimentation est donc maintenant commandable à distance, nous pouvons alors, logiquement, modifier ses sorties par l'envoie des commandes SCPI.

Le déroulement d'un cycle est en deux phases : une phase de charge et une phase de décharge. Donc, il va nous falloir délivrer une tension positive à ses bornes afin de permettre son rechargement.

Le test que nous allons pouvoir exécuter consiste donc à faire délivrer 60 V par l'alimentation. Pour ça, la commande SCPI « VOLT 60 » est supposée répondre à nos attentes.
Le programme est en annexe (Annexe V)

2.3. Interface homme-machine

Afin de permettre « facilement » à un quelconque utilisateur humain d'interagir optimalement avec l'appareil et de répondre à un besoin industriel ou de recherche, il est nécessaire de configurer une couche intermédiaire entre lui et la machine. Cette couche, appelée « IHM » pour Interface Homme-Machine, doit remplir plusieurs critères définis par le client.

Il est important de noter que les profils chargés seront de la forme :

Temps (s)	Courant (A)	Tension Min/Max	Courant Min/Max	Température Min/Max
t_1	$I(t_1)$	V_{min1}/V_{max1}	I_{min1}/I_{max1}	T_{min1}/T_{max1}
t_2	$I(t_2)$	V_{min2}/V_{max2}	I_{min2}/I_{max2}	T_{min2}/T_{max2}
…	…	…	…	…

Comme nous n'avons pas accès à une batterie ni à des relevés capteurs associés, aucune mesure ne peut être affichée. De plus, comme le programme de communication avec l'alimentation n'est pas opérationnel, le chargement d'un profil n'est pas utile. Nous allons donc utiliser une suite de valeurs (que des 0 pour le mode Auto et des relevés aléatoires pour le mode Manuel) afin d'être au moins capable de : différencier les deux modes, charger un fichier csv., et afficher des données chargées en temporel.

Afin d'éviter les conflits de bibliothèques entre PyQt et les versions de Numpy (dûs, d'après nous, à une mise à jour de PyQt au mois de novembre qui peut ne pas encore être compatible totalement avec MacOS), nous avons utilisé la bibliothèque tkinter.

L'IHM se présente ainsi :

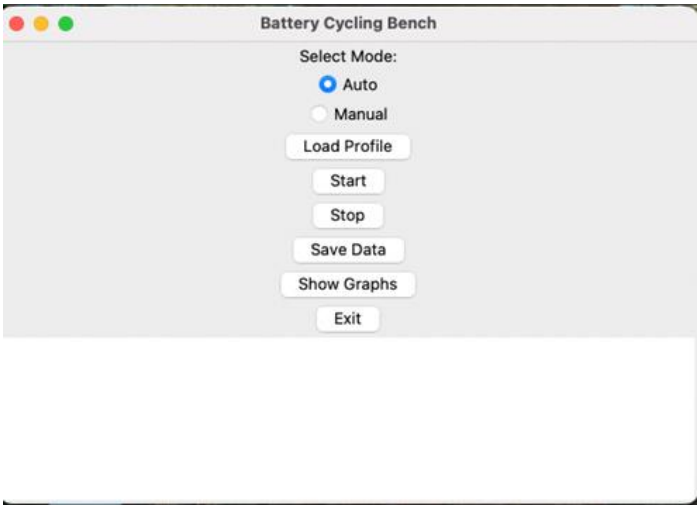


Figure 19. Affichage de l'IHM

Le bouton « Load Profile » permet d'aller chercher les relevés à afficher. Le bouton « Start » lance une simulation d'acquisition de données dans la partie blanche de l'affichage qui s'arrête au pressage du bouton « Stop » dans le cas du mode « Manual » ou seul après 10s dans le cas du mode « Auto ». Ces données (Tensions, Courants, Températures, Défauts BMS, Températures cellules, Tensions cellules) peuvent ensuite être sauvegardées dans un fichier csv. par le bouton « Save Data ». Concernant les graphiques, le mode Auto affiche automatiquement les courbes après 10s (après son arrêt de mesure), le mode Manuel nécessite de cliquer sur « Show Graphs » pour les afficher.

L'appuie sur le bouton « Load Profile » ouvre donc votre fenêtre de recherche de fichiers :

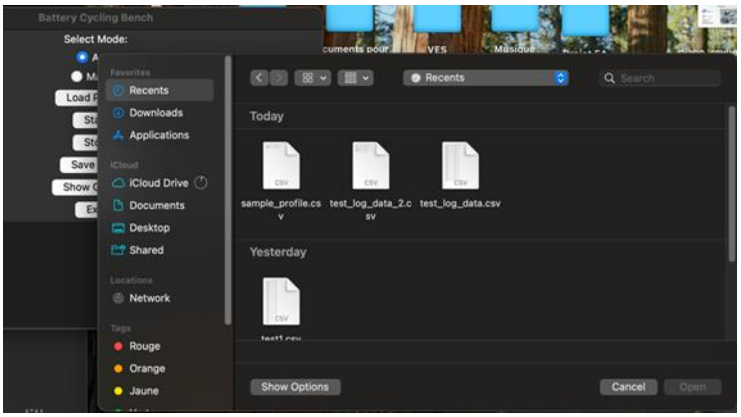


Figure 20. Page de recherche de fichiers

Le profil chargé est arbitrairement le suivant :

sample_profile

Time	Voltage	Current	Temperature	BMS Faults	Cell Voltage	Cell Temperature
1736943899.5500178	3.7	0.5	25	No faults	3.7	25
1736943899.8000193	3.8	0.6	26	No faults	3.8	26
1736943900.0500205	3.9	0.7	27	No faults	3.9	27
1736943900.300021	4.0	0.8	28	No faults	4.0	28
1736943900.5500214	4.1	0.9	29	No faults	4.1	29

Figure 21. Profil de charge aléatoire

Le temps est en seconde, en effet, si on fait la différence entre les décimales des deux premières valeurs : $5500178 - 8000193 = 25000015$. Soit 250ms d'échantillonnage comme indiqué par le cahier des charges.

Une fois, le profil chargé correctement, un message s'affiche :

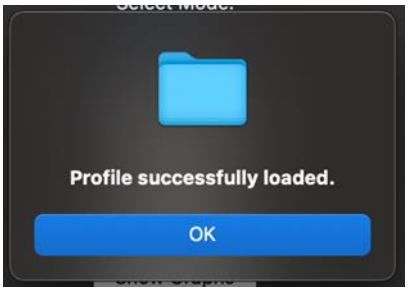


Figure 22. Message de validation du profil

On peut maintenant « Start » l'acquisition des données :

Figure 23. Acquisition de données

L'acquisition commence par « Starting auto mode... » en raison du mode « Auto » sélectionné au départ. On enchaine ensuite sur « Data acquired : » suivi des données acquises.

Après 10s :

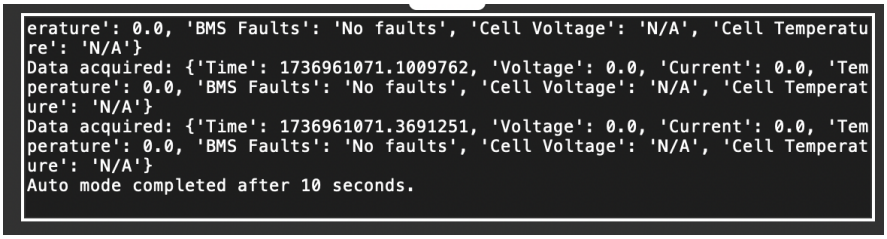


Figure 24. Affichage fin d'acquisition

L'acquisition est finie, il nous est alors possible de télécharger les données simulées dans un fichier csv. avec « Save Data » :

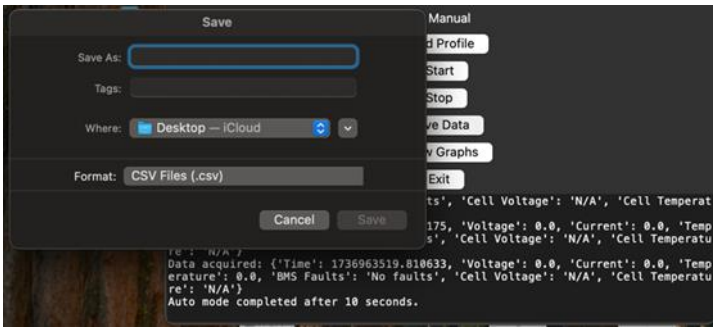


Figure 25. Téléchargement des données acquises

Vous pouvez choisir le nom et la destination du fichier. le document sauvegardé est de la forme Du côté de l'affichage :

test_log_data

Time	Voltage	Current	Temperature	BMS Faults	Cell Voltage	Cell Temperature
1736944700.4800599	0.0	0.0	0.0	No faults	N/A	N/A
1736944700.7431412	0.0	0.0	0.0	No faults	N/A	N/A
1736944701.006367	0.0	0.0	0.0	No faults	N/A	N/A
1736944701.2706041	0.0	0.0	0.0	No faults	N/A	N/A
1736944701.5335522	0.0	0.0	0.0	No faults	N/A	N/A
1736944701.797615	0.0	0.0	0.0	No faults	N/A	N/A
1736944702.064926	0.0	0.0	0.0	No faults	N/A	N/A
1736944702.3264241	0.0	0.0	0.0	No faults	N/A	N/A
1736944702.588158	0.0	0.0	0.0	No faults	N/A	N/A
1736944702.854678	0.0	0.0	0.0	No faults	N/A	N/A
1736944703.11753	0.0	0.0	0.0	No faults	N/A	N/A
1736944703.37917	0.0	0.0	0.0	No faults	N/A	N/A
1736944703.6473448	0.0	0.0	0.0	No faults	N/A	N/A
1736944703.914461	0.0	0.0	0.0	No faults	N/A	N/A
1736944704.180397	0.0	0.0	0.0	No faults	N/A	N/A
1736944704.444083	0.0	0.0	0.0	No faults	N/A	N/A
1736944704.708022	0.0	0.0	0.0	No faults	N/A	N/A
1736944704.966641	0.0	0.0	0.0	No faults	N/A	N/A
1736944705.23283	0.0	0.0	0.0	No faults	N/A	N/A
1736944705.499626	0.0	0.0	0.0	No faults	N/A	N/A
1736944705.764823	0.0	0.0	0.0	No faults	N/A	N/A
1736944706.0272589	0.0	0.0	0.0	No faults	N/A	N/A
1736944706.293134	0.0	0.0	0.0	No faults	N/A	N/A
1736944706.556108	0.0	0.0	0.0	No faults	N/A	N/A

Figure 26.

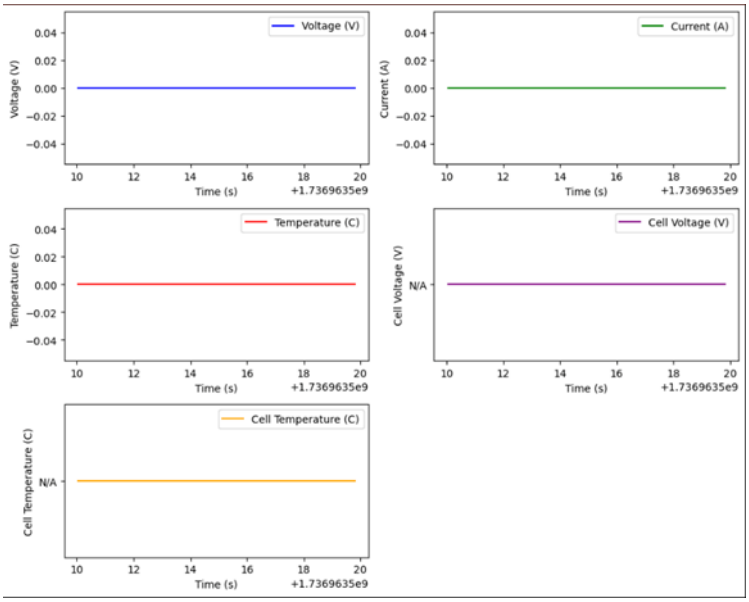


Figure 27. Graphique généré par l'IHM en mode auto

Comme prédit, les relevés sont tous nuls puisque nous avons choisi le mode auto et que nous n'avons pas accès à de véritables mesures de quelconques capteurs. Cependant, les 5 mesures demandées dans le cahier des charges, apparaissent bien : Tensions, Courants, Températures, Défauts BMS, Températures cellules, Tensions cellules.

Si l'on sélectionne le mode Manuel :

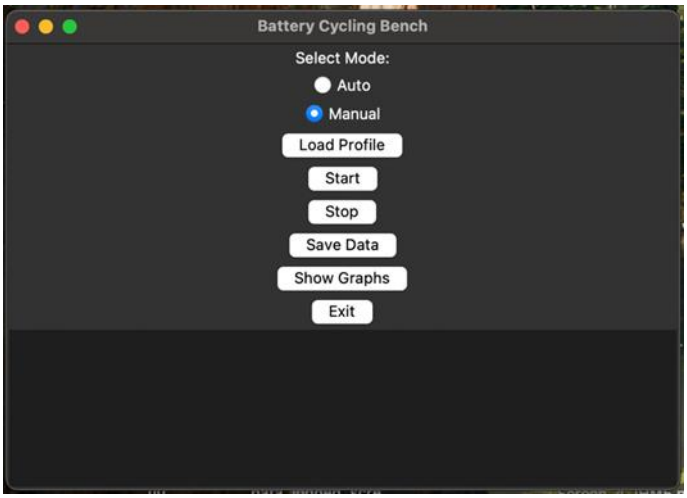


Figure 28. Sélection du mode manuel

L'IHM nécessite le chargement d'un profil de charge/décharge. L'appuie sur le bouton « Start » sans avoir chargé au préalable un profil, provoquera l'affichage d'une fenêtre d'information (idem pour le mode « Auto ») :

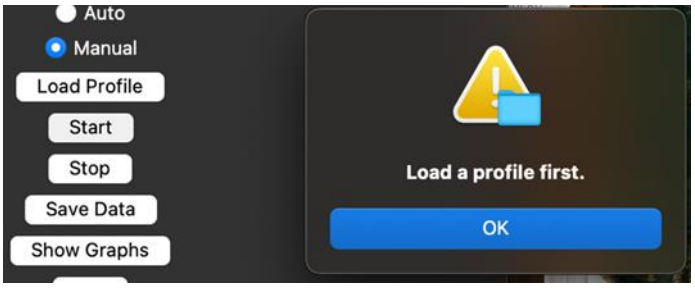


Figure 29. Fenêtre d'indication de nécessité de charger un profil

Le même profile est chargé, nous appuyons identiquement sur « Start » :

Figure 30. Acquisition de données en mode manuel

L'acquisition démarre avec « Starting manual mode... », puis le relevé des données commence.
Elles s'arrêteront à la fin de l'acquisition des données ou à l'appui du bouton « Stop ». Le bouton « Save Data » fait exactement la même chose qu'en mode « Auto » :

test_log_data_2

Time	Voltage	Current	Temperature	BMS Faults	Cell Voltage	Cell Temperature
1736945010.8357792	3.7	0.5	25.0	No faults	3.7	25
1736945011.098391	3.8	0.6	26.0	No faults	3.8	26
1736945011.359682	3.9	0.7	27.0	No faults	3.9	27
1736945011.6179972	4.0	0.8	28.0	No faults	4.0	28
1736945011.8858812	4.1	0.9	29.0	No faults	4.1	29
1736945095.9189382	3.7	0.5	25.0	No faults	3.7	25
1736945096.184544	3.8	0.6	26.0	No faults	3.8	26
1736945096.455424	3.9	0.7	27.0	No faults	3.9	27
1736945096.7129211	4.0	0.8	28.0	No faults	4.0	28
1736945096.980629	4.1	0.9	29.0	No faults	4.1	29

L'observation des graphiques correspondant aux données acquises nécessite que l'on appuie sur « Show Graphs » :

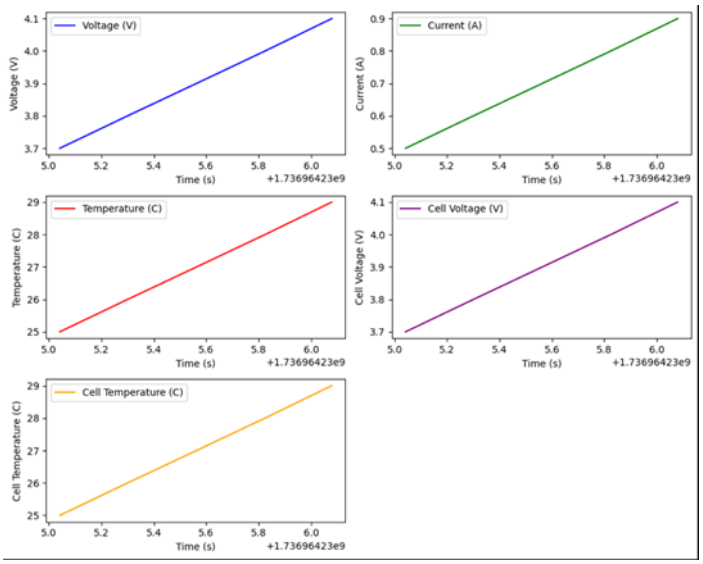


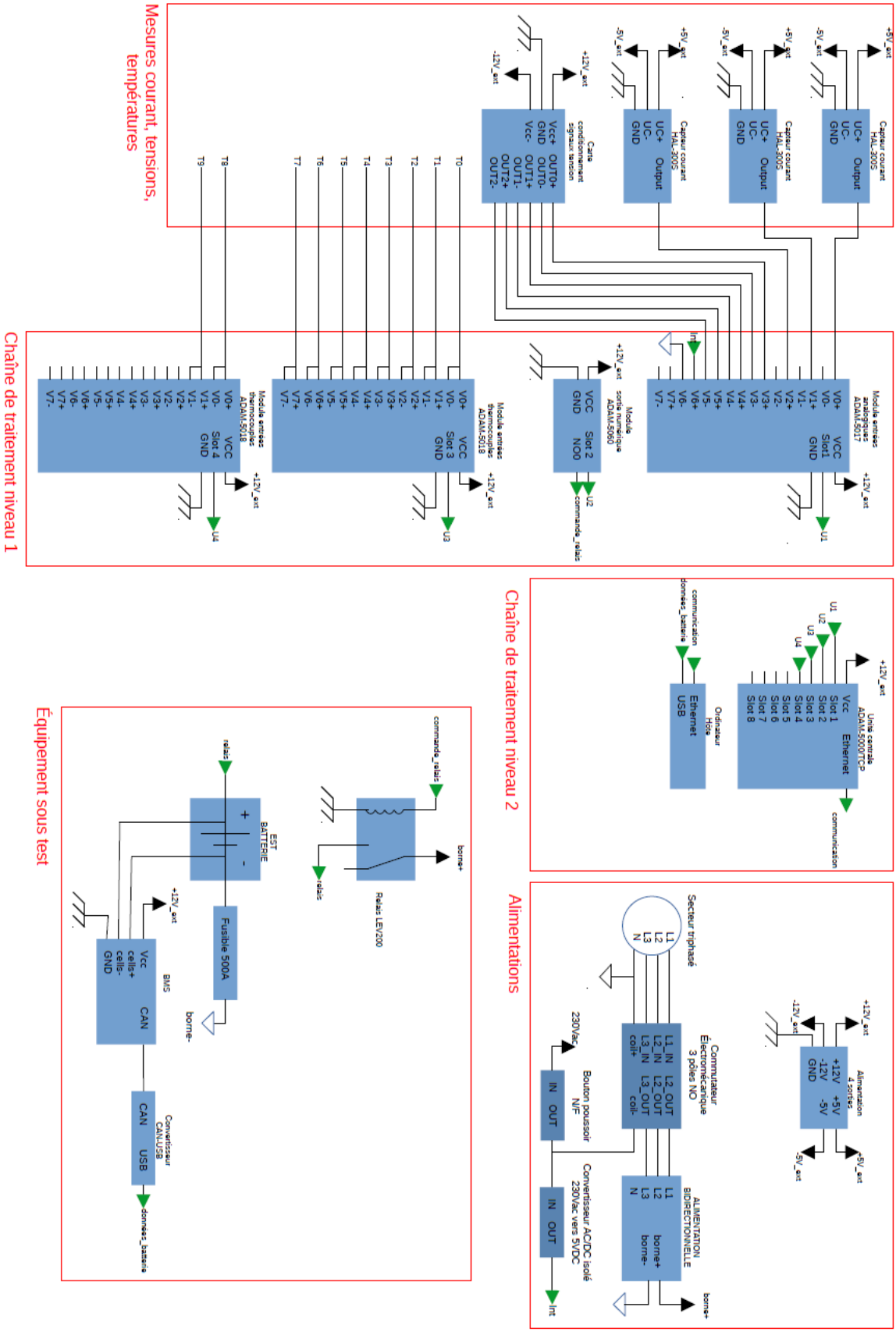
Figure 31. Affichage des données relevées en mode manuel

Les données qui apparaissent ici sont celles du profil chargé en l'absence de réelles données de mesures.

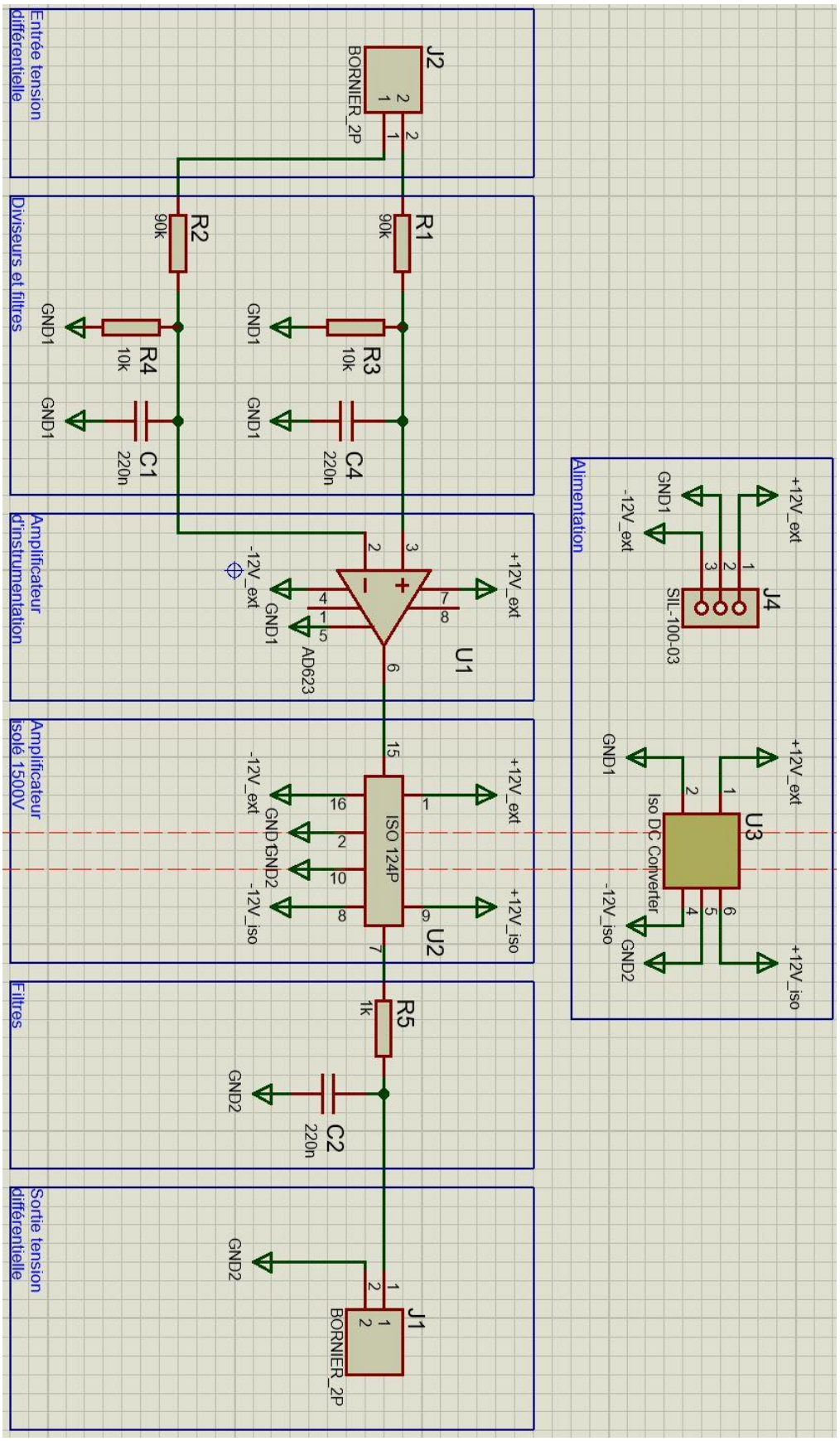
L'étape suivante aurait été de lier le programme définitif de communication avec l'alimentation au programme de l'IHM pour lancer et stopper les cycles de charges/décharges. Ensuite, de configurer les capteurs et l'IHM pour construire des fichiers csv. des données mesurées pour être ensuite capable de les afficher.

Annexes

Annexe I : Schéma électrique de l'architecture interne



Annexe II : Schéma carte d'instrumentation pour la mesure de tension



Annexe III : Référencement des composants de la carte d'instrumentation

Modèle	Nom	commande RS	quantité	prix (€)
INA111AP	AOP instrumentation	197-7135	1	14,86
ISO124P	AOP isolé	300-8243P	1	23,83
TracoPower	Régulateur isolé	311-4871	1	8,84
	Bornier 2 pins	189-5966	5	12,05
	Résistance 90k	707-7824	10	1,48
	Résistance 10k	707-7745	10	1,97
	Résistance 1k	707-8221	10	1,97
	Condensateur 200n	191-3020	5	2,67
	Bornier 3 pins	790-1092	5	5,21
			Total	72,88

Annexe IV : Programme de décodage des trames de données CAN

```
import random
import struct
import time  # Import nécessaire pour ajouter des délais

class CANMessage:
    """Classe pour simuler un message CAN."""
    def __init__(self, arbitration_id, data, transmit_time_ms):
        self.arbitration_id = arbitration_id
        self.data = data
        self.transmit_time_ms = transmit_time_ms  # Temps entre deux transmissions

def simulate_can_messages():
    """Simuler des messages CAN basés sur les données de l'Excel."""
    messages = [
        # Tension totale de la batterie (0x1B0)
        CANMessage(0x1B0, [0x00, 0xC3, 0x50, 0xA0, 0x03, 0x00, 0x05, 0xF0], 1000),
        # Courant de la batterie (0x2B0)
        CANMessage(0x2B0, [0x00, 0x00, 0x00, 0x64, 0x00, 0x00, 0x00, 0x00], 100),
        # Température des cellules (0x3B0)
        CANMessage(0x3B0, [0x00, 0x64, 0x00, 0x32, 0x00, 0x19, 0x00, 0x00], 1000),
    ]
    return random.choice(messages)

def decode_message(message):
    """Décoder les trames CAN en fonction des spécifications de l'Excel."""
    if message.arbitration_id == 0x1B0:
        # Décodage des tensions
        battery_voltage = int.from_bytes(message.data[0:4], "big") * 0.001   # Résolution
0.001
        min_voltage = int.from_bytes(message.data[4:6], "big") * 0.001
        max_voltage = int.from_bytes(message.data[6:8], "big") * 0.001
        print(f"Tension totale : {battery_voltage:.3f} V, Min : {min_voltage:.3f} V, Max :
{max_voltage:.3f} V")
```

```
    elif message.arbitration_id == 0x2B0:
        # Décodage du courant
        current = struct.unpack(">f", bytearray(message.data[0:4]))[0]  # Résolution float
        print(f"Courant de la batterie : {current:.3f} A")
    elif message.arbitration_id == 0x3B0:
        # Décodage des températures
        min_temp = int.from_bytes(message.data[0:2], "big", signed=True) * 0.1  # Résolution
0.1
        max_temp = int.from_bytes(message.data[2:4], "big", signed=True) * 0.1
        avg_temp = int.from_bytes(message.data[4:6], "big", signed=True) * 0.1
        print(f"Température Min : {min_temp:.1f} °C, Max : {max_temp:.1f} °C, Moyenne :
{avg_temp:.1f} °C")
    else:
        print(f"Trame non reconnue : ID={hex(message.arbitration_id)}, Data={message.data}")

def main():
    print("Simulation des trames CAN...")
    try:
        while True:
            # Simuler une trame CAN
            message = simulate_can_messages()
            print(f"Message reçu : ID={hex(message.arbitration_id)}, Data={message.data}")
            decode_message(message)
            # Attendre en fonction du temps de transmission spécifié
            time.sleep(message.transmit_time_ms / 1000.0)  # Convertir ms en secondes
    except KeyboardInterrupt:
        print("Simulation arrêtée.")

if __name__ == "__main__":
    main()
```

Annexe V : Programme de la communication avec l'alimentation

```
import socket
import time

# Paramètres de connexion
IP_ADDRESS = "192.168.0.3"
PORT = 5025
NUM_CYCLES = 5  # Nombre de cycles charge/décharge

# Fonctions utilitaires
def send_command(sock, command):
    """Envoie une commande SCPI via une connexion existante."""
    try:
        if sock.fileno() == -1:  # Vérifie si le socket est actif
            print("Le socket est fermé. Reconnexion...")
            sock = reconnect(sock)

        command = command.strip() + "\n"
        sock.sendall(command.encode())
        time.sleep(1)  # Augmentez le délai à 1 seconde
        response = sock.recv(1024).decode().strip()
        print(f"Commande: {command.strip()}\nRéponse: {response}")
        return response
```

```
    except BrokenPipeError:
        print("Connexion rompue. Tentative de reconnexion...")
        raise
    except Exception as e:
        print(f"Erreur lors de l'envoi de la commande {command.strip()}: {e}")
        raise

def connect_to_device():
    """Établit une connexion au dispositif."""
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(10)
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_KEEPALIVE, 1)  # Active TCP Keep-Alive
        sock.connect((IP_ADDRESS, PORT))
        print("Connecté à l'alimentation.")
        return sock
    except Exception as e:
        print(f"Erreur lors de la connexion : {e}")
        raise

def reconnect(sock):
    """Rétablit une connexion en cas de perte."""
    try:
        sock.close()
    except:
        pass
    return connect_to_device()

def log_device_error(sock):
    """Récupère les erreurs de l'appareil."""
    try:
        error_log = send_command(sock, "SYST:ERR?")
        print(f"Log de l'appareil : {error_log}")
    except Exception as e:
        print(f"Impossible de lire le log de l'appareil : {e}")

def initialize_device(sock):
    """Initialise le dispositif et active le contrôle à distance."""
    # Identification de l'appareil
    response = send_command(sock, "*IDN?")
    if not response:
        raise Exception("Aucune réponse de l'appareil. Vérifiez la connexion.")

    # Activation du contrôle à distance et configuration
    send_command(sock, "SYST:LOCK ON")
    send_command(sock, "SYST:TIMEOUT 0")  # Désactivation du timeout SCPI
    send_command(sock, "*CLS")  # Efface les erreurs précédentes
    send_command(sock, "*RST")  # Réinitialisation de l'appareil

def cycle_charge_discharge(sock):
    """Effectue les cycles de charge et décharge."""
    for cycle in range(1, NUM_CYCLES + 1):
        print(f"\nDébut du cycle {cycle}/{NUM_CYCLES}")

        # Étape 1 : Charge
        print("Démarrage de la charge...")
        try:
```

```
            time.sleep(0.5)  # Délai avant de configurer la tension
            send_command(sock, "VOLT 10")
            time.sleep(10)  # Temps de charge (exemple : 60 secondes)

        except BrokenPipeError:
            print("Reconnexion nécessaire pendant la charge.")
            sock = reconnect(sock)
            log_device_error(sock)
            continue

        # Étape 2 : Décharge
        #print("Démarrage de la décharge...")
        #try:
            #send_command(sock, "SINK:CURR 2")  # Exemple : Courant de décharge négatif
            #time.sleep(10)  # Temps de décharge (exemple : 60 secondes)
        #except BrokenPipeError:
            #print("Reconnexion nécessaire pendant la décharge.")
           #sock = reconnect(sock)
            #log_device_error(sock)
            #continue

    print("\nTous les cycles sont terminés.")
    send_command(sock, "OUTP:STAT OFF")  # Désactive la sortie

# Script principal
try:
    sock = connect_to_device()
    initialize_device(sock)
    send_command(sock, "OUTP ON")
    cycle_charge_discharge(sock)
except BrokenPipeError:
    print("Erreur critique : Connexion rompue. Tentative de réinitialisation...")
    sock = reconnect(sock)
except Exception as e:
    print(f"Erreur critique : {e}")
finally:
    if sock:
        try:
            send_command(sock, "OUTP OFF")
            send_command(sock,  "OUTP:STAT  OFF")   #  Assurez-vous  que  l'alimentation  est
arrêtée

        except:
            pass
        sock.close()
        print("Connexion fermée.")
```

Annexe VI : Programme de l'interface homme-machine

```
import tkinter as tk
from tkinter import filedialog, messagebox
import csv
import time
from threading import Thread
import matplotlib.pyplot as plt
```

```
class BatteryCyclingApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Battery Cycling Bench")

        # Variables
        self.data_log = []
        self.is_running = False
        self.profile_data = []
        self.mode = tk.StringVar(value="auto")

        # UI Components
        self.create_widgets()

    def create_widgets(self):
        # Mode Selection
        tk.Label(self.root, text="Select Mode:").pack()
        tk.Radiobutton(self.root, text="Auto", variable=self.mode, value="auto").pack()
        tk.Radiobutton(self.root, text="Manual", variable=self.mode, value="manual").pack()

        # Buttons
        tk.Button(self.root, text="Load Profile", command=self.load_profile).pack()
        tk.Button(self.root, text="Start", command=self.start_cycling).pack()
        tk.Button(self.root, text="Stop", command=self.stop_cycling).pack()
        tk.Button(self.root, text="Save Data", command=self.save_data).pack()
        tk.Button(self.root, text="Show Graphs", command=self.show_graphs).pack()
        tk.Button(self.root, text="Exit", command=self.root.quit).pack()

        # Log Display
        self.log_text = tk.Text(self.root, state='disabled', height=10)
        self.log_text.pack()

    def load_profile(self):
        file_path = filedialog.askopenfilename(filetypes=[("CSV Files", "*.csv")])
        if not file_path:
            return

        with open(file_path, newline='') as csvfile:
            reader = csv.DictReader(csvfile)
            self.profile_data = [row for row in reader]

        messagebox.showinfo("Profile Loaded", "Profile successfully loaded.")

    def start_cycling(self):
        if not self.profile_data:
            messagebox.showwarning("Warning", "Load a profile first.")
            return

        self.is_running = True
        if self.mode.get() == "auto":
            self.log_message("Starting auto mode...")
            Thread(target=self.run_auto_mode).start()
        elif self.mode.get() == "manual":
            self.log_message("Starting manual mode...")
            Thread(target=self.run_manual_mode).start()

    def stop_cycling(self):
```

```
        self.is_running = False
        self.log_message("Cycling stopped.")

    def run_auto_mode(self):
        start_time = time.time()
        while self.is_running and (time.time() - start_time) < 10:
            self.acquire_data()
            time.sleep(0.25)
        self.is_running = False
        self.log_message("Auto mode completed after 10 seconds.")
        self.show_graphs()

    def run_manual_mode(self):
        for profile in self.profile_data:
            if not self.is_running:
                break
            self.acquire_data(profile)
            time.sleep(0.25)

    def acquire_data(self, profile=None):
        # Simulated data acquisition
        if profile:
            data = {
                "Time": time.time(),
                "Voltage": float(profile["Voltage"]),
                "Current": float(profile["Current"]),
                "Temperature": float(profile["Temperature"]),
                "BMS Faults": profile.get("BMS Faults", "No faults"),
                "Cell Voltage": profile.get("Cell Voltage", "N/A"),
                "Cell Temperature": profile.get("Cell Temperature", "N/A"),
            }
        else:
            data = {
                "Time": time.time(),
                "Voltage": 0.0,
                "Current": 0.0,
                "Temperature": 0.0,
                "BMS Faults": "No faults",
                "Cell Voltage": "N/A",
                "Cell Temperature": "N/A",
            }

        self.data_log.append(data)
        self.log_message(f"Data acquired: {data}")

    def log_message(self, message):
        self.log_text.config(state='normal')
        self.log_text.insert(tk.END, f"{message}\n")
        self.log_text.config(state='disabled')
        self.log_text.see(tk.END)

    def save_data(self):
        if not self.data_log:
            messagebox.showwarning("Warning", "No data to save.")
            return
```

```
        file_path = filedialog.asksaveasfilename(defaultextension=".csv", filetypes=[("CSV
Files", "*.csv")])
        if not file_path:
            return

        with open(file_path, mode='w', newline='') as csvfile:
            fieldnames = ["Time", "Voltage", "Current", "Temperature", "BMS Faults", "Cell
Voltage", "Cell Temperature"]
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            writer.writerows(self.data_log)

        messagebox.showinfo("Data Saved", "Data successfully saved.")

    def show_graphs(self):
        if not self.data_log:
            messagebox.showwarning("Warning", "No data to show.")
            return

        times = [data["Time"] for data in self.data_log]
        voltages = [data["Voltage"] for data in self.data_log]
        currents = [data["Current"] for data in self.data_log]
        temperatures = [data["Temperature"] for data in self.data_log]
        cell_voltages = [data["Cell Voltage"] for data in self.data_log]
        cell_temperatures = [data["Cell Temperature"] for data in self.data_log]

        plt.figure(figsize=(10, 8))

        # Voltage graph
        plt.subplot(3, 2, 1)
        plt.plot(times, voltages, label="Voltage (V)", color='blue')
        plt.xlabel("Time (s)")
        plt.ylabel("Voltage (V)")
        plt.legend()

        # Current graph
        plt.subplot(3, 2, 2)
        plt.plot(times, currents, label="Current (A)", color='green')
        plt.xlabel("Time (s)")
        plt.ylabel("Current (A)")
        plt.legend()

        # Temperature graph
        plt.subplot(3, 2, 3)
        plt.plot(times, temperatures, label="Temperature (C)", color='red')
        plt.xlabel("Time (s)")
        plt.ylabel("Temperature (C)")
        plt.legend()

        # Cell Voltage graph
        plt.subplot(3, 2, 4)
        plt.plot(times, cell_voltages, label="Cell Voltage (V)", color='purple')
        plt.xlabel("Time (s)")
        plt.ylabel("Cell Voltage (V)")
        plt.legend()

        # Cell Temperature graph
```

```
        plt.subplot(3, 2, 5)
        plt.plot(times, cell_temperatures, label="Cell Temperature (C)", color='orange')
        plt.xlabel("Time (s)")
        plt.ylabel("Cell Temperature (C)")
        plt.legend()

        plt.tight_layout()
        plt.show()

if __name__ == "__main__":
    root = tk.Tk()
    app = BatteryCyclingApp(root)
    root.mainloop()
```

Annexe VII : Référencement des composants choisis

Description	Fournisseur	Prix unitaire	Quantité	Prix total	Lien
Modules E/S 8-slot	MOUSER	462,65 €	1	462,65 €	https://www.mouser.fr/ProductDetail/Advantech/ADAM-5000-TCP-CE?qs=rrS6PyfT74cf3LBoWKahdA%3D%3D&srsltid=AfmBOornNhDIzxewDiTJBCwd1zBnbl4S1nvflt2VcHvQUHvN2_TptOPm
Module E/S 8-Ch analogic	MOUSER	191,90 €	1	191,90 €	https://www.mouser.fr/ProductDetail/Advantech/ADAM-5017-A4E?qs=rrS6PyfT74egohykrOjXPQ%3D%3D&srsltid=AfmBOopZjEQdQFEs7YGAMaQxXnQ3J9EbVhiA8ojHxbcJgnGJ4RqJqVfC
Modules E/S 7-ch Thermocouple Input	MOUSER	260,30 €	2	520,60 €	https://www.mouser.fr/ProductDetail/Advantech/ADAM-5018P-AE?qs=dawwGz1r7uaGcrgsk0BnOw%3D%3D&srsltid=AfmBOooGodjmfS6nqJ3sWyvZfnuSRXXK1vokQi1GZZ09HRHS5ODZ94ki
Module, 6-Ch Relay Output	Advantech	125,42 €	1	125,42 €	https://buy.advantech.eu/I-O-Devices-Communication/Remote-I-O-Modules-Modularized-I-O-Systems-I-O-Modules/model-ADAM-5069-AE.htm
Capteur de courant	Digikey	31,81 €	3	95,43 €	https://www.digikey.fr/fr/products/detail/lem-usa-inc/HAS-300-S/1026543?srsltid=AfmBOorYJ47fnfpOtalMlrl9TmOqUFl7tjUNFj1ECU4I3dfjI07RZPns
Type T, à raccord dénudé	RS	14,27 €	10	142,70 €	https://fr.rs-online.com/web/p/thermocouples/2713144?searchId=3b46ea42-861e-486d-84e3-db6ee6e22875&gb=s
Alimentation à découpage, 65W	RS	118,63 €	1	118,63 €	https://fr.rs-online.com/web/p/alimentations-a-decoupage/2003522
Alimentation à découpage, 80W	RS	44,56 €	1	44,56 €	https://fr.rs-online.com/web/p/alimentations-a-decoupage/2006222
Contacteur électromécanique 30A	MOUSER	354,26 €	1	354,26 €	https://www.mouser.fr/ProductDetail/Siemens/LEN00C003277B?qs=rSMjJ%252B1ewcS1H08WAIw%2FBA%3D%3D
Arrêt urgence bouton switch	MOUSER	41,66 €	1	41,66 €	https://www.mouser.fr/ProductDetail/E-Switch/E100-R11-H1KR?qs=mELouGlnn3earQph70Z2Tg%3D%3D
AC/DC convertisseur 10W, 5 V	MOUSER	20,90 €	1	20,90 €	https://www.mouser.fr/ProductDetail/Murata-Power-Solutions/BAC10S05DC?qs=IKkN%2F947nfAzeOJoi9kdrw%3D%3D

Convertisseur USB vers CAN Innomaker	RS	75,75 €	1	75,75 €	https://fr.rs-online.com/web/p/hat-et-complements-raspberry-pi/2526692