



UNIVERSITÉ DE  
MONTPELLIER

# Rapport de développeur Python et C++



Service HFA

Julien-Servel Thomas

Vitesco Technologies

BUT2 IUT Montpellier-Sète

Jérôme Godia

Fabien Michel

20/01-28/03 2025

## Remerciements

Je tiens à exprimer ma sincère gratitude à toutes les personnes qui m'ont accompagné et soutenu tout au long de mon stage chez Vitesco Technologies.

Tout d'abord, je remercie chaleureusement **Jérôme Godia**, mon tuteur de stage, pour son accompagnement et sa disponibilité. Son expertise dans le domaine ainsi que sa patience m'ont permis de m'intégrer rapidement au sein de l'équipe et d'acquérir de nouvelles compétences techniques. Jérôme a su me guider dans l'ensemble de mes missions, en m'offrant un cadre propice à l'apprentissage et en me permettant de participer activement aux différents projets. Grâce à lui, j'ai pu découvrir les défis d'un projet de grande envergure et comprendre les attentes d'un client à travers des objectifs techniques concrets.

Je souhaite également remercier **Damien François**, **Théo Morin**, et **Stéphane Rey** pour leur soutien et leur bienveillance. Leur ouverture et leur partage d'expérience m'ont permis d'approfondir mes connaissances et d'appréhender les différents secteurs au sein de l'entreprise. Grâce à leur aide, j'ai pu élargir ma compréhension du travail en entreprise, des exigences clients et des processus de développement. Chacun a contribué, à sa manière, à m'intégrer dans l'équipe et à me faire découvrir le fonctionnement d'une entreprise à taille humaine, tout en me permettant de voir l'impact concret de nos actions sur les projets en cours.

Merci à tous pour l'opportunité d'avoir pu vivre cette expérience enrichissante.

## Résumé :

Ce document est un rapport de stage effectué au sein de l'entreprise Vitesco Technologies, rédigé par un étudiant de l'IUT de Montpellier dans le cadre de la fin de sa deuxième année d'études. Il présente le travail réalisé pendant la période de stage, principalement axé sur le développement de solutions de test et de gestion d'alimentations électroniques dans le domaine des capteurs pour véhicules.

Le rapport aborde trois missions principales :

1. **Développement d'une interface homme-machine (IHM) en Python** pour piloter une alimentation Keysight via une communication LAN, dans le but de tester et de valider le bon fonctionnement des capteurs dans des conditions spécifiques. Cette partie implique des compétences en programmation Python et en gestion de communication avec des équipements électroniques.
2. **Création d'une IHM pour la gestion d'une alimentation sur des durées longues (plus de 24h)**, afin d'assurer la stabilité et la continuité des tests sur des périodes étendues, tout en intégrant des fonctions de surveillance et de contrôle en temps réel.
3. **Réalisation d'un projet basé sur Arduino**, permettant de développer une solution de contrôle électronique pour l'évaluation de capteurs, en mettant l'accent sur l'optimisation et la fiabilité des tests dans un environnement d'essai.

Le travail effectué durant ce stage a permis de découvrir les enjeux techniques d'un projet d'envergure pour un client, tout en mettant en pratique des compétences en développement logiciel et en gestion de projets électroniques.

Mots-clés : Vitesco Technologies, développement, Python, Arduino, IHM, alimentation Keysight, capteurs, véhicules, tests fonctionnels, communication LAN, gestion d'alimentation, interface.

Sommaire :

## Table des matières

Remerciements.....	2
Résumé :.....	3
1-Introduction.....	5
2-Contexte du stage.....	6
2.1-Présentation de l'entreprise.....	6
2.2-Dimension commerciale.....	6
2.3-Enjeux du stage.....	7
3-Analyse et spécifications techniques.....	8
3.1 Objectifs du stage.....	8
3.2 Analyse technique.....	11
3.3 Cahier des charges.....	13
4- Rapport technique.....	16
4.1 Projet 1.....	16
4.1.1 Présentation.....	16
4.1.1 Outils et technologies utilisées.....	16
4.1.2 Développement du projet.....	16
4.1.3 Problèmes rencontrés et solutions apportées.....	21
4.1.4 Résultats et bilan technique.....	22
4.2 Projet 2.....	23
4.2.1 Présentation et enjeux du projet.....	23
4.2.2. Développement du projet.....	23
4.2.3 Problèmes rencontrés et solutions apportées.....	27
4.2.4 Résultats et bilan technique.....	27
4.3 Projet 3.....	28
4.3.1 Présentation et enjeux du projet.....	28
4.3.2 Outils et technologies utilisées.....	28
4.3.3 Développement du projet.....	29
Explication.....	31
Explication.....	33
4.3.4 Problèmes rencontrés et solutions apportées.....	34
4.3.5 Résultats et bilan technique.....	34
5-Conclusion.....	35
5.1 Bilan du travail réalisé.....	35
5.2 Compétences acquises.....	35
5.3 Perspectives.....	36
5.4 Bilan de l'expérience en entreprise.....	36
5.5 Bilan de l'expérience personnelle.....	37
6 - Annexes :.....	38
6.1 Glossaire.....	38
6.2 Index des figures.....	39

## 1-Introduction

Ce rapport présente le travail que j'ai réalisé au sein de l'entreprise Vitesco Technologies, du 20 janvier au 28 mars 2025, dans le cadre de ma formation à l'IUT Informatique de Montpellier. Vitesco Technologies est une entreprise spécialisée dans la fourniture de solutions technologiques pour l'industrie automobile, avec un focus particulier sur les capteurs et la gestion des alimentations électroniques. Dans ce contexte, mon stage m'a permis d'acquérir des compétences pratiques dans le développement d'interfaces homme-machine (IHM) et la gestion des tests sur des capteurs électroniques.

L'objectif principal de mon stage était de contribuer à la mise en place et à l'optimisation de systèmes de test pour les capteurs de véhicules, en utilisant des plateformes telles qu'Arduino et Python. Les tâches auxquelles j'ai participé ont couvert le développement d'IHM pour contrôler des alimentations, la gestion des essais sur des bancs de test, ainsi que l'optimisation des processus de tests de capteurs en environnement réel.

Ce rapport sera structuré en trois grandes parties. Tout d'abord, dans un premier temps, je présenterai l'entreprise Vitesco Technologies et les enjeux techniques du stage, en décrivant l'environnement de travail et les objectifs qui m'ont été fixés. Ensuite, je détaillerai les missions effectuées, à savoir le développement d'IHM sous Python, l'optimisation de tests sur Arduino, et la gestion des alimentations dans un environnement de tests. Enfin, je conclurai ce rapport en analysant les compétences acquises, les résultats obtenus, et les perspectives possibles pour la suite de mon parcours professionnel.

## 2-Contexte du stage

### 2.1-Présentation de l'entreprise

Vitesco Technologies est une entreprise internationale spécialisée dans la conception et la fourniture de solutions innovantes pour l'industrie automobile. Issue de la scission de Continental en 2018, l'entreprise se concentre principalement sur les technologies liées à l'électrification des véhicules et aux systèmes de contrôle électroniques. Vitesco Technologies est présente à l'échelle mondiale, avec des sites de production et des centres de R&D répartis dans plusieurs pays.

La société propose une large gamme de produits, allant des solutions d'alimentation électrique et de gestion thermique, aux capteurs sophistiqués utilisés pour des applications variées telles que la gestion des émissions et des performances des moteurs, la sécurité, et l'automatisation des véhicules.

Le siège social de Vitesco Technologies est situé en Allemagne, mais l'entreprise dispose également de nombreuses filiales à travers le monde, notamment en Europe, en Asie et en Amérique du Nord. Le secteur d'activité est en pleine croissance, porté par l'essor des véhicules électriques et des technologies de conduite autonome. L'entreprise compte plusieurs milliers de collaborateurs et regroupe des ingénieurs spécialisés dans les systèmes électroniques et logiciels embarqués.

Mon stage s'est déroulé au sein de l'équipe chargée de la gestion des capteurs pour l'industrie automobile, une équipe dont l'objectif principal est d'assurer le bon fonctionnement des capteurs utilisés dans les véhicules en effectuant des tests et des validations dans des conditions réelles. Cette équipe est composée d'ingénieurs, de techniciens, ainsi que de plusieurs autres collaborateurs spécialisés dans la conception de solutions matérielles et logicielles.

### 2.2-Dimension commerciale

Vitesco Technologies propose une large palette de produits et services destinés aux constructeurs automobiles. Parmi les produits phares, on retrouve des solutions de gestion de l'alimentation, des modules de contrôle des émissions, ainsi que des capteurs à haute précision. L'entreprise développe des capteurs utilisés dans des applications critiques, notamment les capteurs de mouvement et de position utilisés pour l'automatisation des véhicules et les systèmes d'assistance à la conduite.

Les clients de Vitesco Technologies sont principalement des grands acteurs de l'industrie automobile, tels que des constructeurs de véhicules de première monte et des équipementiers. L'entreprise intervient également dans des projets pour des clients dans le secteur de l'automobile haute performance, ainsi que dans des secteurs annexes liés à l'électrification des transports.

Le marché sur lequel opère Vitesco Technologies est particulièrement dynamique, car il répond à des besoins croissants en matière d'innovation technologique dans le secteur automobile, notamment en lien avec la transition énergétique, l'électrification des véhicules, et la conduite autonome. Ces défis exigent une forte capacité d'innovation et des tests rigoureux pour garantir la performance, la sécurité et la fiabilité des systèmes développés.

## 2.3-Enjeux du stage

L'objectif principal de mon stage chez Vitesco Technologies était d'acquérir une expérience pratique dans la gestion des tests de capteurs électroniques dans un contexte automobile. Au-delà de la simple manipulation des capteurs, il était essentiel de comprendre les besoins techniques et les défis auxquels l'équipe est confrontée, tout en contribuant à améliorer l'efficacité des tests et l'interfaçage des différents outils.

Les besoins techniques étaient multiples, notamment dans la mise en place de tests pour les capteurs en conditions réelles sur des bancs de tests mécaniques et sur des véhicules. Il s'agissait de veiller à la précision des mesures, à la qualité des données collectées, mais aussi d'optimiser le traitement des informations et leur analyse. Les travaux réalisés ont inclus le développement de solutions IHM pour interagir avec les équipements de test, le développement de scripts en Python pour automatiser certaines tâches, ainsi que la gestion de l'intégration des capteurs avec des systèmes plus complexes.

Les projets auxquels j'ai participé ont donc été à la fois techniques et organisationnels, permettant de répondre à des besoins spécifiques en termes de performance des capteurs et d'optimisation des processus de test dans un environnement automobile.

## 3-Analyse et spécifications techniques

### 3.1 Objectifs du stage

L'objectif principal de mon stage chez **Vitesco Technologies** était de développer et d'optimiser des outils logiciels et matériels destinés à la gestion et à l'automatisation des tests de capteurs automobiles. Dans un contexte où les capteurs jouent un rôle clé dans la modernisation et l'électrification des véhicules, il est crucial de mettre en place des systèmes de test fiables, précis et automatisés.

Mon stage s'est déroulé au sein de l'équipe **HFA (Hands-Free Access)**, spécialisée dans le développement et le test de capteurs d'accès sans contact. L'enjeu était de concevoir des solutions permettant une meilleure automatisation pour la gestion de l'alimentation des capteurs via des interface pour simplifier les test.

Pour répondre à ces besoins, mon travail s'est structuré autour de **trois grands projets**, chacun ayant pour objectif d'améliorer l'efficacité des tests et la communication entre les systèmes.

#### **Projet 1 : Développement d'une IHM en Python pour piloter une alimentation en fonction de commutateurs Arduino**

**Objectif** : Automatiser la gestion de l'alimentation des capteurs en fonction de certaines conditions détectées par un Arduino, tout en permettant un suivi en temps réel via une interface graphique intuitive.

##### **Enjeux :**

- Permettre aux ingénieurs d'adapter dynamiquement l'alimentation des capteurs sans intervention manuelle.
- Assurer une communication fluide entre l'Arduino et l'alimentation programmable.
- Concevoir une IHM ergonomique permettant d'afficher l'état des commutateurs et la tension appliquée.

#### **Projet 2 : Conception d'une IHM pour la gestion d'une alimentation programmable sur des durées prolongées**



**Objectif** : Développer un outil permettant aux ingénieurs de définir des cycles d'alimentation automatisés (exemple : 2V pendant 30 min, puis 4V pendant 20 min, avant l'arrêt automatique).

**Enjeux** :

- Offrir une solution simple et configurable pour planifier des séquences de tension sans intervention humaine.
- Assurer la fiabilité du programme sur de longues durées (>24h).
- Automatiser la transition entre les différentes phases du test grâce à un chronomètre intégré.

### **Projet 3 : Interface entre un capteur et un PC via un Arduino**

**Objectif** : Mettre en place une passerelle entre un capteur et un PC via un microcontrôleur (Arduino), en utilisant le **protocole LIN** ou un **monitoring série**.

**Enjeux** :

- Assurer une transmission fiable des données du capteur vers un PC.
- Étudier et choisir le protocole de communication le plus adapté.
- Intégrer une solution modulaire pouvant être réutilisée pour différents capteurs.

#### **Un enjeu commun : automatisation et optimisation des tests**

Ces trois projets répondaient à un besoin crucial au sein de l'équipe : améliorer l'efficacité des tests en réduisant l'intervention humaine et en optimisant l'interaction entre les équipements. En automatisant certaines tâches et en développant des interfaces intuitives, l'objectif était d'offrir aux ingénieurs :

- Un gain de temps dans la mise en place des tests.
- Une meilleure fiabilité des résultats en limitant les erreurs manuelles.
- Une standardisation des procédures, facilitant la reproductibilité des tests.

#### **Compétences mises en œuvre**

Ce stage a été une opportunité d'approfondir plusieurs compétences techniques et méthodologiques, notamment :

Programmation logicielle : Développement en Python (IHM, gestion des alimentations) et en C++/Arduino pour la communication avec les capteurs.

Interfaçage matériel : Utilisation d'un Arduino comme pont entre des capteurs et un PC, pilotage d'une alimentation via LAN/USB.

Protocole de communication : Étude et mise en œuvre du LIN et du monitoring série pour la récupération de données du capteur.

Collaboration et méthodologie : Échange avec les ingénieurs ou techniciens pour adapter les solutions aux besoins réels du projet.

Ces acquis m'ont permis d'avoir une vision plus claire des **contraintes industrielles**, tout en me confrontant aux défis du développement logiciel et de l'intégration électronique dans un environnement de test automobile.

Dans la section suivante, je vais analyser en détail les spécifications techniques et les solutions mises en place pour chacun de ces projets.

## 3.2 Analyse technique

### Projet 1 : Développement d'une IHM pour piloter une alimentation en fonction de switchs contrôlées via Arduino

#### Problématique et contexte

Dans le cadre des tests de capteurs, il est parfois nécessaire de modifier dynamiquement l'alimentation d'un circuit en fonction d'entrées spécifiques (température par exemple). L'objectif de ce projet était donc de concevoir une IHM en Python capable de piloter une alimentation programmable en fonction de l'état des switch détectés par un Arduino.

#### Développement et solutions techniques

- **Communication entre l'Arduino et l'ordinateur** : J'ai utilisé une liaison USB pour transmettre l'état des commutateurs depuis l'Arduino vers le programme Python.
- **Contrôle de l'alimentation programmable** : Le programme Python envoie des commandes via un protocole LAN ou USB à l'alimentation pour modifier la tension en fonction des entrées reçues.
- **Conception de l'IHM** : Une interface graphique intuitive a été développée en utilisant QtDesigner, permettant de visualiser l'état des commutateurs et la tension appliquée en temps réel.

#### Résultats :

Grâce à cette solution, les tests des capteurs peuvent être réalisés plus rapidement et avec une meilleure précision, sans nécessiter d'intervention manuelle pour ajuster l'alimentation, tout pourra ce faire automatiquement.

### Projet 2 : IHM de gestion d'alimentation avec chronomètre intégré

#### Problématique et contexte

Certains tests nécessitent une alimentation variable sur une période prolongée. Par exemple, il peut être nécessaire d'alimenter un capteur à 2V pendant 30 minutes, puis à 4V pendant 20 minutes avant de couper l'alimentation. L'objectif de ce projet était de développer une IHM permettant de gérer ces cycles automatiquement.

#### Développement et solutions techniques

- **Programmation de l'alimentation programmable** : L'IHM envoie des commandes à l'alimentation en suivant un scénario défini (tension et durée).
- **Interface graphique** : L'IHM permet à l'utilisateur de définir les valeurs de tension et les durées correspondantes via des champs configurables.

- **Gestion du temps** : Un chronomètre intégré gère les transitions automatiques entre les différentes tensions définies, et permet de n'avoir aucun technicien sur place afin de chronométrer le temps.

### Résultats

Ce projet a permis d'automatiser les tests nécessitant des variations de tension, réduisant ainsi la charge de travail des opérateurs et améliorant la répétabilité des tests.

## Projet 3 : Interface entre un capteur et un PC via Arduino

### Problématique et contexte

Dans certains cas, les capteurs utilisés pour les tests ne peuvent pas être directement connectés à un PC. Il est donc nécessaire d'utiliser un microcontrôleur (Arduino) comme interface pour lire les données du capteur et les transmettre à l'ordinateur via un protocole de communication adapté.

### Développement et solutions techniques

- **Choix du protocole de communication** : Selon le capteur, j'ai étudié l'utilisation du **protocole LIN** ou d'un **monitoring série** pour récupérer les données.
- **Programmation de l'Arduino** : Le microcontrôleur a été programmé pour recevoir les données du capteur et les transmettre au PC via USB.
- **Traitement des données sur l'ordinateur** : Un programme Python a été développé pour lire, afficher et enregistrer les valeurs reçues.

### Résultats et bénéfices

Ce projet a permis de rendre les données du capteur accessibles depuis un PC, facilitant ainsi leur exploitation et leur analyse. Il offre également une solution modulaire pouvant être adaptée à différents types de capteurs.

### 3.3 Cahier des charges

Le cahier des charges a défini les objectifs, les exigences et les contraintes techniques des trois projets réalisés durant mon stage. Ces projets avaient pour but d'améliorer la gestion des tests en automatisant le contrôle des alimentations et en facilitant l'exploitation des données capteurs.

#### **Projet 1 : IHM pour piloter une alimentation en fonction de switches contrôlés via Arduino**

##### **Objectif**

Développer une interface permettant de contrôler une alimentation programmable en fonction de l'état de switches détectés par un Arduino. L'IHM doit afficher l'état des entrées et ajuster la tension de l'alimentation en conséquence.

##### **Exigences fonctionnelles**

- Affichage en temps réel de l'état des commutateurs et de la tension appliquée.
- Transmission des états des commutateurs de l'Arduino vers le PC via une liaison série.
- Modification automatique de la tension de l'alimentation en fonction des entrées reçues.
- Interface ergonomique pour une utilisation simple et rapide.
- Mode autonome capable de gérer l'alimentation et l'Arduino en cas de déconnexion (coupure de courant par exemple)

##### **Exigences techniques**

- Utilisation d'un protocole de communication USB pour relier l'Arduino au PC.
- Envoi de commandes à l'alimentation via une connexion LAN ou USB.
- Développement de l'IHM en Python avec QtDesigner.

##### **Contraintes**

- Temps de réponse rapide pour assurer une mise à jour quasi instantanée.
- Fonctionnement stable sur de longues sessions de test.
- Compatibilité avec les équipements existants dans l'entreprise.

## **Projet 2 : IHM pour automatiser la gestion d'une alimentation avec cycles de tension**

### **Objectif**

Créer une IHM permettant de définir et d'exécuter des changements de tension sur une alimentation programmable. L'interface doit gérer des cycles d'alimentation sur des périodes prolongées et afficher un chronomètre pour suivre la progression des tests.

### **Exigences fonctionnelles**

- Permettre à l'utilisateur de définir deux étapes, UB et UB\_Min avec une tension et une durée associée.
- Exécuter automatiquement les changements de tension selon le scénario configuré.
- Afficher un chronomètre en temps réel pour suivre l'évolution du cycle.
- Bouton stop qui clignote pour faciliter la reconnaissance du test qui est finis.

### **Exigences techniques**

- Envoi de commandes à l'alimentation via LAN/USB pour modifier la tension.
- Gestion du temps en Python pour assurer des transitions précises entre les phases.
- Développement d'une IHM claire et intuitive permettant de modifier facilement les valeurs.

### **Contraintes**

- Précision du chronomètre pour garantir une durée exacte des tests.
- Fonctionnement continu sur une période pouvant dépasser 24 heures.
- Stabilité du programme pour éviter toute interruption en cours de test.

### **Projet 3 : Interface entre un capteur et un PC via Arduino**

#### **Objectif**

Mettre en place une solution permettant d'interfacer un capteur automobile avec un PC via un Arduino. L'interface doit assurer la récupération des données du capteur et leur transmission en temps réel vers l'ordinateur.

#### **Exigences fonctionnelles**

- Lire et transmettre les données brutes du capteur vers le PC.
- Choisir un protocole de communication adapté (LIN ou monitoring série).
- Afficher en temps réel les valeurs mesurées et permettre l'enregistrement des données.

#### **Exigences techniques**

- Programmation de l'Arduino pour gérer la lecture et la transmission des données.
- Mise en place d'une liaison USB ou LIN entre l'Arduino et le capteur.
- Développement d'un programme en Python pour traiter et visualiser les données reçues.

#### **Contraintes**

- Stabilité de la communication pour éviter les pertes de données.
- Compatibilité avec plusieurs types de capteurs utilisés dans l'entreprise.
- Rapidité d'exécution pour garantir une acquisition en temps réel.

## 4- Rapport technique

### 4.1 Projet 1

#### 4.1.1 Présentation

Ce projet répondait à un **besoin spécifique** rencontré par l'équipe de test des capteurs chez Vitesco Technologies : lors des tests, il était nécessaire d'ajuster dynamiquement la tension appliquée aux capteurs en fonction d'un certain état physique représenté par des commutateurs. Ces réglages étant effectués **manuellement**, ils étaient sujets à des erreurs humaines et demandaient du temps.

L'objectif était donc de **rendre ce processus plus fiable et automatisé**, en permettant aux ingénieurs de définir la tension souhaitée pour chaque état et de laisser le programme s'occuper des ajustements en temps réel.

#### 4.1.1 Outils et technologies utilisées

Afin de répondre aux exigences du projet, plusieurs outils et bibliothèques ont été utilisés :

##### Langages de programmation

- **Python** pour le développement de l'application principale et l'interface utilisateur.
- **C++ (Arduino)** pour la détection des commutateurs et la transmission des données au PC.

##### Environnements de développement

- **Visual Studio Code** pour la programmation Python.
- **Arduino IDE** pour la programmation et la gestion du microcontrôleur.

##### Bibliothèques et frameworks

- **PyQt6** pour la conception de l'interface graphique.
- **QtDesigner** pour la création rapide de l'IHM (qui ressemble à Scene Builder de java).
- **pySerial** pour établir la communication série entre l'Arduino et le PC.
- **PyVISA** pour l'envoi de commandes vers l'alimentation programmable via USB ou LAN.

#### 4.1.2 Développement du projet

##### 4.1.2.1 Création de l'interface graphique en Python

La première étape du développement a été la conception d'une **interface** pour permettre aux ingénieurs de sélectionner facilement les tensions à appliquer.



Ayant déjà utilisé SceneBuilder en Java, j'ai recherché une alternative équivalente pour Python et découvert **QtDesigner**, qui offre une approche similaire en glisser-déposer avec les mêmes outils comme les VBox, Hbox, labels etc... L'IHM a été pensée pour faciliter la configuration des tensions et afficher en temps réel l'état des commutateurs activés.

L'IHM est composée de plusieurs éléments interactifs :

- **Des champs de saisie** permettant de configurer la tension associée à chaque commutateur.
- **Un affichage en temps réel** montrant quel commutateur est activé.
- **Des boutons de connexion** pour établir la communication avec l'Arduino et l'alimentation.
- **Un journal de log** affichant l'historique des actions et les changements de switch avec le temps.

```
self.label_switch1 = QtWidgets.QLabel(parent=self.centralwidget)
self.label_switch1.setObjectName("label_switch1")
self.horizontalLayout_5.addWidget(self.label_switch1)
self.checkBox_switch1 = QtWidgets.QCheckBox(parent=self.centralwidget)
self.checkBox_switch1.setText("")
self.checkBox_switch1.setObjectName("checkBox_switch1")
self.horizontalLayout_5.addWidget(self.checkBox_switch1)
self.lineEdit_Switch1 = QtWidgets.QLineEdit(parent=self.centralwidget)
self.lineEdit_Switch1.setObjectName("lineEdit_Switch1")
```

Figure 1: Utilisation de PyQt6 pour les champs de saisie

Ce code initialise l'interface et définit les différents widgets permettant à l'utilisateur de configurer les tensions et de surveiller l'état du système.

## Explication

- Chaque **widget** de l'interface est identifié par un nom unique afin de pouvoir être manipulé dans le code.
- Un **événement est déclenché** dès qu'une valeur est modifiée par l'utilisateur, ce qui donne une mise à jour dynamique.
- L'interface a été pensée pour **éviter les erreurs de saisie**, notamment en limitant les valeurs acceptées.

```
# Define a validator to accept only numbers (float)
float_validator = QRegularExpressionValidator(QRegularExpression(r"^[+-]?[d*\\.\\d+$"))
self.ui.lineEdit_Switch1.setValidator(float_validator)
self.ui.lineEdit_Switch2.setValidator(float_validator)
self.ui.lineEdit_Switch3.setValidator(float_validator)
self.ui.lineEdit_Switch4.setValidator(float_validator)
self.ui.lineEdit_Ampere.setValidator(float_validator)
```

Figure 2: Validation uniquement float pour les valeur des volts et ampere

#### 4.1.2.2 Programmation de l'Arduino et communication série

L'Arduino est un rôle principal dans ce projet en assurant la détection des commutateurs et l'envoi des informations vers le PC. Sans lui on ne peut pas savoir si un commutateurs est activé ou non.

##### Principe de fonctionnement :

- Chaque commutateur est relié à une broche en entrée avec **pull-up** activé (HIGH par défaut).
- Lorsqu'un commutateur est activé, il est connecté à la **masse** et passe à LOW.
- L'Arduino envoie un **message** au PC, contenant le numéro du commutateur activé.

```
Serial.print("Switch:");
Serial.println(i + 1); // Les switches restent numérotés 1-4
```

Figure 3: Envoie du switch depuis l'arduino

```
def read_from_port(self):
    global connection
    while self.running:
        if self.serial_connection and self.serial_connection.is_open:
            try:
                message = self.serial_connection.readline().decode('utf-8').strip()
                if message:
                    self.message_received.emit(message)
                    connection = True
            except Exception as e:
                connection = False
                print(f"Erreur lors de la lecture du port série : {e}")
                break
```

Figure 4: Reception du message de l'arduino via python

## Explication

- L'utilisation du pull-up interne évite d'avoir besoin de résistances externes. Ce qui as faciliter le montage et câblage de l'Arduino
- Chaque fois qu'un **changement d'état** est détecté, l'Arduino envoie un message sous la forme : « Switch:X »
- L'application Python reçoit ces données et les traite.

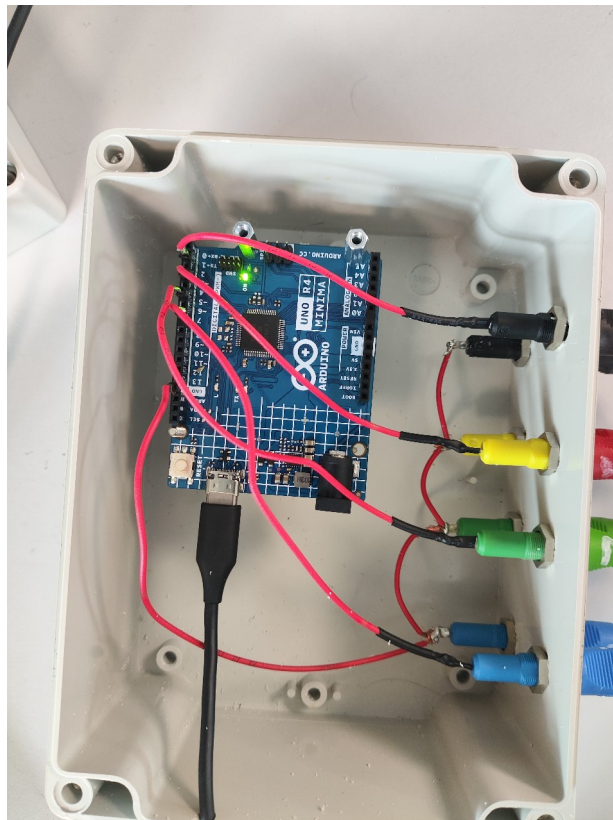


Figure 5: Montage Arduino du Projet 1

### 4.1.2.3 Interfaçage avec l'alimentation programmable

Une fois la communication entre l'Arduino et l'IHM établie, il restait à contrôler l'alimentation programmable.

L'alimentation devait recevoir des **commandes spécifiques** pour modifier sa tension en fonction du switch activé. Ces commandes sont trouvable sur leur site web en téléchargeant la documentation de l'alimentation.

```
# 2. Si LAN échoue, chercher en USB
if not is_power_connected:
    try:
        resources = rm.list_resources()
        # Filtre pour Keysight (USB Vendor ID 0x2A8D)
        usb_resources = [res for res in resources if "USB" in res and "0x2A8D" in res]
        print(usb_resources)

        if usb_resources:
            # Prend la première ressource USB trouvée
            instr = rm.open_resource(usb_resources[0], timeout=1000)
            idn = instr.query("*IDN?")
            instr.close()
            print(idn)

            if "E36103B" in idn:
                adrs_alimentation = usb_resources[0]
                is_power_connected = True

    except (pyvisa.errors.VisaIOError, IndexError):
        print("rien en USB")
        pass # Aucune alim USB trouvée
```

Figure 6: Détection de l'alimentation en USB

```
def update_voltage(instrument_address, voltage):
    """
    connect to the instrument and change the voltage
    """
    try:
        rm = pyvisa.ResourceManager()

        print(f"Tentative de connexion à l'instrument à l'adresse {instrument_address}...")
        instrument = rm.open_resource(instrument_address)
        print("Connexion réussie à l'instrument.")

        instrument.write(f"VOLT {voltage}")
        print(f"Tension réglée à {voltage}V.")

        instrument.write('OUTP ON')

    except pyvisa.VisaIOError as e:
        print(f"Erreur de communication avec l'instrument : {e}")
    finally:
        try:
            instrument.close()
            print("Connexion à l'instrument fermée.")
        except NameError:
            print("L'instrument n'a pas été ouvert, aucune connexion à fermer.")
```

Figure 7: Fonction du changement de voltage de l'alimentation

## Explication

- PyVISA détecte l'alimentation connectée via USB/LAN et garde en mémoire l'adresse de l'alimentation (USB ou LAN).
- À chaque activation de commutateur, le programme lit la valeur configurée dans l'IHM et met à jour la tension de l'alimentation.
- Un **système de logs** permet de surveiller les valeurs appliquées et de détecter d'éventuelles anomalies.

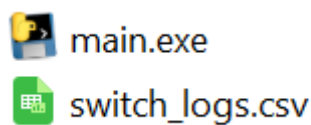


Figure 8: Fichier de log projet 1 v

### 4.1.3 Problèmes rencontrés et solutions apportées

Durant le développement, plusieurs problèmes ont dû être résolus :

#### Problème 1 : Instabilité de la connexion LAN

- L'alimentation saturait lorsque trop de requêtes étaient envoyées en mode LAN et décidait de ne plus répondre au commande ce qui empêchais la communication.
- **Solution** : Passage à la connexion **USB**, plus stable et fiable.

#### Problème 2 : Gestion automatique des connexions

- Le programme supposait qu'un utilisateur était toujours présent pour relancer les connexions, reconnecter, résoudre les problèmes de voltage etc...
- **Solution** : Ajout d'un **mode autonome** qui détecte et reconnecte automatiquement l'Arduino et l'alimentation en cas de déconnexion et vérifie la tension.

#### Problème 3 : Mauvaise gestion après une longue période

- L'IHM ralentissait après plusieurs heures ou bien arrêtait de fonctionner et ne répondait plus. Le chronomètre s'arrêtait, les switch n'était pas update.
- **Solution** : Optimisation du programme en gérant mieux l'actualisation des valeurs et la communication série, refactoring de certaines fonctions.

#### 4.1.4 Résultats et bilan technique

##### Fonctionnalités finales développées :

Une IHM intuitive permettant de sélectionner la tension de chaque commutateur. Une communication stable et automatique entre l'Arduino vers l'alimentation en passant par l'ordinateur. Un système de gestion d'erreurs garantissant la continuité des tests que ce soit avec un humain ou non.

##### Compétences acquises :

- Programmation en **Python** et **Arduino** plus avancées.
- Gestion de **problèmes et erreurs**.
- Refactorisation de code.

Ce projet a permis d'**automatiser un processus manuel**, rendant les tests plus rapides et plus fiables chez Vitesco Technologies.

## 4.2 Projet 2

### 4.2.1 Présentation et enjeux du projet

Ce deuxième projet s'inscrit dans la continuité du premier mais avec un objectif différent. Il s'agissait cette fois-ci de contrôler l'alimentation en lui faisant suivre une séquence automatique de tensions selon une durée prédéfinie.

Le but était de remplacer une intervention manuelle répétitive, source d'erreurs et chronophage, par un programme assurant la transition automatique entre deux tensions (UB et UB\_MIN).

L'IHM devait donc permettre à l'utilisateur de :

- **Définir les deux tensions** (UB et UB\_MIN).
- **Choisir l'ordre d'application** (commencer par UB ou UB\_MIN).
- **Définir une durée X pour chaque tension.**
- **Afficher un chronomètre en temps réel** montrant la progression du test.
- **Activer un bouton STOP rouge clignotant** à la fin du test.

### Outils et technologies utilisées

Langages de programmation

- **Python** pour la conception de l'interface et la gestion des tests.

Environnements de développement

- **Visual Studio Code** pour la programmation Python.

Bibliothèques et frameworks

- **PyQt6** pour la conception de l'interface graphique.
- **QtDesigner** pour créer l'IHM plus rapidement.
- **PyVISA** pour l'envoi des commandes à l'alimentation programmable via USB.

### 4.2.2. Développement du projet

#### 4.2.2.1 Adaptation de l'interface graphique (IHM)

L'interface devait être simple et intuitive, avec une configuration des tensions et durées facile

**Éléments de l'IHM :**

- Deux **champs de saisie** pour entrer UB et UB\_MIN.
- Un **sélecteur d'ordre** permettant de choisir si le test commence par UB ou UB\_MIN.
- Un **selectionneur de durée** pour chaque phase.
- Un **chronomètre** affichant le temps restant pour chaque tension.
- Un **bouton STOP rouge clignotant** s'activant une fois le test terminé.
- Un **bouton pour connecter** l'alimentation

```
# Validator for line edits
float_validator = QRegularExpressionValidator(QRegularExpression(r"^[+-]?\\d*\\.\\d+$"))
self.ui.lineEdit_ub.setValidator(float_validator)
self.ui.lineEdit_ub_min.setValidator(float_validator)
```

Figure 9: Validation des valeurs saisie pour UB et UB\_Min

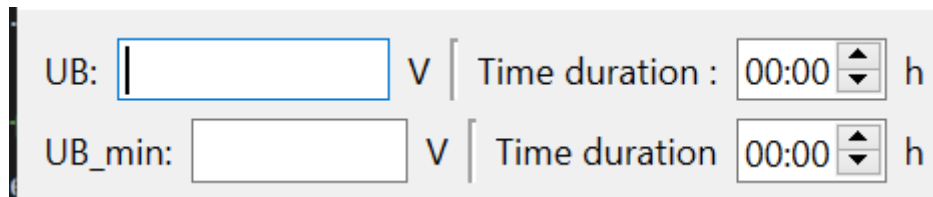


Figure 10: Première partie de l'ihm du projet 2

```
if self.ui.checkBox_UB1.isChecked():
    try:
        value1 = float(self.ui.lineEdit_ub.text())
    except ValueError:
        show_popup("Veuillez entrer une valeur numérique valide pour UB1")
        return
    # Récupérer la durée à partir du QTimeEdit (ici timeEdit_ub)
    qtime1 = self.ui.timeEdit_ub.time() # QTime
    duration1 = qtime1.hour() * 3600 + qtime1.minute() * 60 + qtime1.second()
elif self.ui.checkBox_UB_min1.isChecked():
    try:
        value1 = float(self.ui.lineEdit_ub_min.text())
    except ValueError:
        show_popup("Veuillez entrer une valeur numérique valide pour UB_min1")
        return
    qtime1 = self.ui.timeEdit_ub_min.time() # QTime
    duration1 = qtime1.hour() * 3600 + qtime1.minute() * 60 + qtime1.second()
else:
    show_popup("Veuillez sélectionner une option pour la mesure 1")
    return
```

Figure 11: Start du programme du projet 2 avec les checkboxes

## Explication

- Les **valeurs saisies** sont validées pour éviter les erreurs (on évite de mettre la tension à « é » au lieu de 2 pas exemple).
- Le **sélecteur d'ordre** sont juste des case à cocher pour permettre une compréhension simple.
- L'utilisateur doit sélectionner une durée sur une horloge, qui sera ensuite utilisée pour la transition entre les tensions.



#### 4.2.2.2 Programmation du chronomètre et exécution du cycle de test

L'objectif était d'envoyer **une première tension pendant un temps donné**, puis de basculer automatiquement vers la **seconde tension**, avant de **terminer le test** et d'activer le bouton STOP.

##### Mécanisme du chronomètre :

1. Au lancement du test, l'IHM envoie **UB** ou **UB\_MIN** à l'alimentation, selon l'ordre choisi.
2. Un **chronomètre démarre** et décrémente la durée définie.
3. Une fois le temps écoulé, l'alimentation bascule vers la **seconde tension**.
4. Le second chronomètre démarre pour compter la durée restante.
5. Une fois le test terminé, **le bouton STOP clignotant s'active**.

```
def start_countdown(self, duration, label, callback):
    """
    Démarre un compte à rebours pour la durée spécifiée.
    """
    self.current_time = 0 # On commence à 0
    self.countdown_duration = duration # Durée totale
    self.countdown_label = label # UB sélectionné
    self.countdown_callback = callback #prochain UB
    self.countdown_timer.start(1000)

def update_countdown(self):
    """Met à jour le label du compte à rebours chaque seconde."""
    if self.current_time <= self.countdown_duration:
        hours = self.current_time // 3600
        minutes = (self.current_time % 3600) // 60
        seconds = self.current_time % 60
        self.countdown_label.setText(f"{hours:02}:{minutes:02}:{seconds:02}")
        self.current_time += 1
    else:
        self.countdown_timer.stop()
        if self.countdown_callback:
            self.countdown_callback()
```

Figure 12: Code de la gestion des UB et UB\_Min

##### Explication

- L'IHM utilise **un timer qui met à jour l'affichage toutes les secondes**.
- Lorsqu'un timer atteint **zéro**, une commande SCPI est envoyée via **PyVISA** pour modifier la tension.
- L'état actuel est affiché en temps réel sur l'interface.

#### 4.2.2.3 Envoi des commandes SCPI à l'alimentation

L'alimentation devait être contrôlée **via USB**, en utilisant **PyVISA** pour lui envoyer des commandes en **SCPI** (Standard Commands for Programmable Instruments).

**Commandes SCPI utilisées :**

- **VOLTAGE <valeur>** → Définit la tension de sortie.
- **OUTPUT ON/OFF** → Active ou désactive la sortie de l'alimentation.

Comme ce code rejoint celui du projet 1 j'ai juste repris les code des [Figure 6](#) [Figure 7](#). Et les implémenté dans ce projet.

#### Explication

- Le programme détecte l'alimentation connectée et établit une communication via PyVISA.
- Lors du changement de tension, une commande SCPI est envoyée pour modifier la valeur de sortie.

#### 4.2.2.4 Gestion du bouton STOP clignotant

Pour signaler la fin du test, un bouton STOP rouge clignotant devait être affiché.

**Fonctionnement du bouton STOP :**

- Le bouton est invisible tant que le test est en cours.
- Une fois la séquence terminée, il devient rouge et clignote toutes les 500ms.
- Un clic de l'utilisateur désactive le clignotement et réinitialise l'interface.

```
def toggle_stop_blink(self):  
    self.blink_state = not self.blink_state  
    color = "#FF0000" if self.blink_state else "#8B0000"  
    self.ui.btn_stop.setStyleSheet(f"background-color: {color};")
```

Figure 13: Code du bouton stop clignotant

```
# Timer for Stop button blinking  
self.blink_timer = QTimer(self)  
self.blink_timer.timeout.connect(self.toggle_stop_blink)  
self.blink_state = False
```

Figure 14: Management du déclencheur du blinking

#### Explication

- Un **timer séparé** alterne la couleur du bouton pour le rendre bien visible.
- L'utilisateur peut cliquer dessus pour **arrêter l'animation et réinitialiser le test**.

### 4.2.3 Problèmes rencontrés et solutions apportées

#### Problème 1 : Chronomètre imprécis

**Constat** : Parfois, le chronomètre ne s'arrêtait pas exactement à 0 mais 1 ou 2 et affichait des valeurs incorrectes.

**Solution** : Remplacement du timer par un compteur mis à jour automatiquement pour être sûr d'arriver à la bonne valeur.

Il s'agit du seul problème rencontré car le code viens du projet 1 une fois terminer donc la gestion avec l'alimentation est fiable et stable. Et les switch sont juste remplacé par Ub et Ub\_Min donc la logique reste la même. Seul le timer posait problème

### 4.2.4 Résultats et bilan technique

#### Fonctionnalités finales développées :

- ✓ **IHM intuitive** avec saisie des tensions et de la durée.
- ✓ **Cycle de test automatique** avec passage d'UB à UB\_MIN.
- ✓ **Chronomètre en temps réel** affichant la progression.
- ✓ **Gestion des erreurs et validation des saisies.**
- ✓ **Bouton STOP rouge clignotant** indiquant la fin du test.

#### Compétences acquises :

- Programmation avancée de **timers en Python**.
- Utilisation des **commandes SCPI avec PyVISA**.
- Gestion des transitions de tension via **une interface automatisée**.
- Prise en compte des **retours utilisateurs** pour améliorer l'ergonomie.

Ce projet a permis d'automatiser un processus auparavant manuel, garantissant des tests plus fiables et reproductibles chez Vitesco Technologies.

## 4.3 Projet 3

### 4.3.1 Présentation et enjeux du projet

L'objectif de ce projet était d'utiliser un **Arduino Due** comme **passerelle de communication** entre un **capteur** et un **PC**. Contrairement aux projets précédents, qui reposaient sur une simple communication série avec l'alimentation programmable, ce projet impliquait une gestion avancée des protocoles de communication, notamment :

- Un mode **MONITORING**, utilisant une liaison **série standard** pour transmettre les données brutes entre le PC et le capteur.
- Un mode **LIN**, basé sur le protocole **Local Interconnect Network (LIN)**, nécessitant une gestion spécifique des trames et l'envoi d'un **break LIN** pour synchroniser les échanges.

Le projet devait permettre :

- L'envoi de commandes depuis le PC vers le capteur via l'Arduino.
- La réception et la transmission des données capteur vers le PC.
- Un affichage sur un écran LCD pour faciliter le débogage et suivre l'état du système.
- Un passage automatique entre le mode **MONITORING** et le mode **LIN**, selon l'activité détectée sur la liaison série.

L'architecture de communication prévue était la suivante :

1. **PC** → **RX3 (Arduino)** → **TX3 (Capteur)** : Envoi des commandes vers le capteur.
2. **Capteur** → **RX1 (Arduino)** → **TX1 (PC)** : Retour des données vers le PC.
3. **L'Arduino gère la détection du protocole LIN** et bascule automatiquement entre les deux modes.

Le projet a progressé jusqu'à une **gestion fonctionnelle du monitoring et du mode LIN**, mais la **gestion bidirectionnelle complète en LIN n'a pas pu être finalisée** par manque de temps.

### 4.3.2 Outils et technologies utilisées

#### Matériel

- **Arduino Due**, qui dispose de plusieurs ports série matériels (UART).
- **Capteur**, capable de fonctionner en deux modes distincts avec des débits spécifiques.
- **Écran LCD**, utilisé pour afficher des informations de diagnostic et faciliter le débogage.

#### Logiciel et langages

- **Arduino IDE** pour la programmation du microcontrôleur.
- **C++ (Arduino)** pour la gestion des communications série et du protocole LIN.

### 4.3.3 Développement du projet

#### 4.3.3.1 Configuration de l'écran LCD

L'écran LCD a été utilisé pour afficher les informations en temps réel sur l'état de la communication. Avant d'implémenter la logique de gestion des données capteur, la première étape a été de tester l'affichage sur le LCD.

```
#include <LiquidCrystal.h>

const int rs = 44, en = 48, d4 = 53, d5 = 51, d6 = 49, d7 = 47;
const int contrastPin = 6;
const int backlightPin = 10;

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
```

Figure 15: initialisation des broches arduino pour le lcd

```
// Affichage LCD
void updateLCD() {
    static unsigned long lastUpdate = 0;
    if (millis() - lastUpdate < 500) return;
    lastUpdate = millis();

    lcd.setCursor(0, 0);
    lcd.print("LIN:");
    lcd.print(linCounter);

    lcd.setCursor(0, 1);
    lcd.print(currentMode == LIN_MODE ? "LIN " : "MON ");
}
```

Figure 16: code d'affichage sur l'écran lcd

```
void setup() {  
  lcd.begin(16, 2);  
  analogWrite(contrastPin, 100);  
  analogWrite(backlightPin, 255);  
  lcd.print("System Ready");  
  delay(1000);  
  lcd.clear();  
}
```

Figure 17: initialisation du lcd dans le setup de l'arduino

### Explication

- L'écran est connecté aux **broches 44, 48, 53, 51, 49, 47**.
- La **contraste et le rétroéclairage** sont ajustés via **PWM**.
- À l'initialisation, un message "**System Ready**" est affiché.

#### 4.3.3.2 Gestion du mode MONITORING

Le mode MONITORING est le mode **par défaut** dans lequel l'Arduino fonctionne comme un **relais transparent** entre le PC et le capteur.

#### Fonctionnement :

- **Tout message reçu sur RX3 (PC)** est immédiatement transmis sur TX3 (Capteur).
- **Tout message reçu sur RX1 (Capteur)** est immédiatement transmis sur TX1 (PC).

```
void handleMonitoringMode() {  
    // RX3 (PC) -> TX3 (Capteur)  
    while (Serial3.available()) {  
        char received = Serial3.read();  
        if (received != lastRX3) { // Éviter les répétitions  
            Serial3.write(received); // Forward vers le capteur  
            lastActivity = millis();  
            lastRX3 = received; // Sauvegarder pour comparaison  
        }  
    }  
  
    while (Serial3.available()) Serial3.read(); // VIDAGE RX3  
  
    // RX1 (Capteur) -> TX1 (PC)  
    while (Serial1.available()) {  
        char received = Serial1.read();  
        Serial.print("RX1 reçu : "); Serial.println(received, HEX);  
        if (received != lastTX1) { // Éviter les répétitions  
            Serial1.write(received);  
            Serial1.flush();  
            Serial.print("TX1 envoyé : "); Serial.println(received, HEX);  
            lastTX1 = received; // Sauvegarder pour comparaison  
        }  
    }  
  
    while (Serial1.available()) Serial1.read(); // VIDAGE RX1
```

Figure 18: Code de gestion du mode monitoring

### Explication

- Les **données envoyées par le PC** sont directement transférées vers le capteur sans modification.
- Les **réponses du capteur** sont renvoyées au PC.
- Un **système de filtrage** empêche la retransmission de messages identiques pour éviter les boucles infinies.

### Problème détecté :

- Sans gestion des tampons série, certains messages étaient **perdus ou corrompus**.  
**Solution** : Implémentation d'un **vidage des buffers** après chaque lecture.

#### 4.3.3.3 Détection et passage en mode LIN

Le protocole LIN requiert une gestion particulière car il nécessite l'envoi d'un break LIN avant chaque transmission pour synchroniser le capteur.

### Mécanisme de bascule vers le mode LIN :

- L'Arduino surveille **USART3 (RX3)** et détecte un **break LIN** grâce au bit **RXBRK**.
- Si un **break est détecté**, l'Arduino bascule en **LIN\_MODE** et modifie la vitesse de transmission.

```
// Détection du break LIN matériel
void checkLINBreak() {
    if (USART3->US_CSR & US_CSR_RXBRK) {
        USART3->US_CR = US_CR_RSTSTA;
        switchToLIN();
        linCounter++; // Incrémente seulement quand un break est détecté
    }
}
```

Figure 19: Code de détection du LIN beark

```
void switchToLIN() {
    if (currentMode != LIN_MODE) {
        Serial3.end();
        Serial1.end();
        Serial3.begin( );
        Serial1.begin( );
        Serial.print("Mode LIN actif, baud  \n");
        currentMode = LIN_MODE;
        lastActivity = millis();
        breakSent = false;
        while (Serial3.available()) Serial3.read(); // Vider buffer
        while (Serial1.available()) Serial1.read(); // Vider buffer
    }
}
```

Figure 20: code du changement vers le LIN

### Explication

- L'Arduino utilise un **registre matériel** (US\_CSR & US\_CSR\_RXBRK) pour détecter le break LIN.
- Lorsqu'un **break est détecté**, l'Arduino :
  1. **Réinitialise les ports série.**
  2. **Modifie la vitesse de transmission.**
  3. **Affiche "Mode LIN" sur le LCD.**



#### 4.3.3.4 Gestion du mode LIN

Une fois en mode LIN, l'Arduino doit gérer **l'envoi d'un break LIN** avant toute commande.

**Mécanisme d'envoi de break :**

1. Activation du mode **STTBRK** (Start Break).
2. Attente de **2,2 ms** (durée typique du break LIN).
3. Désactivation du mode **STPBRK** (Stop Break).
4. Transmission des données au capteur.

```
// Envoi d'un break LIN
void sendLINBreak() {
    USART3->US_CR = US_CR_STTBRK;
    delayMicroseconds(2200); // Durée du break
    USART3->US_CR = US_CR_STPBRK;
    delayMicroseconds(500); // Pause après le break
}
```

Figure 21: Génération du break LIN

#### Explication

- L'envoi du **break** assure la **synchronisation avec le capteur**.
- Une fois le **break envoyé**, l'Arduino peut transmettre les **données LIN** normalement.

**Problème détecté :**

- Sans une **pause après le break**, certaines commandes étaient mal interprétées.  
**Solution :** Ajout d'un **délai de 500µs** après le break pour stabiliser la communication.

#### 4.3.3.5 Retour en mode MONITORING

Si aucune activité LIN n'est détectée pendant **100 ms**, l'Arduino revient en mode MONITORING.

```
void loop() {  
    checkLINBreak();  
  
    if (currentMode == LIN_MODE) {  
        handleLINMode();  
        if (millis() - lastActivity > LIN_TIMEOUT) switchToMonitoring();  
    } else {  
        handleMonitoringMode();  
    }  
  
    updateLCD();  
}
```

Figure 22: Fonction loop de l'Arduino

### Explication

- Une **temporisation surveille l'absence d'activité** pour éviter une détection erronée.
- Lorsque le délai est dépassé, l'Arduino :
  - **Réinitialise la vitesse de transmission.**
  - **Repassé en mode MONITORING.**

### 4.3.4 Problèmes rencontrés et solutions apportées

#### Problème 1 : Perte de messages en UART

**Constat** : Les messages trop rapides étaient parfois ignorés.

**Solution** : Mise en place d'un **vidage des buffers série** après chaque transmission.

#### Problème 2 : Transition instable entre MONITORING et LIN

**Constat** : Des bascules trop fréquentes perturbaient la communication.

**Solution** : Ajout d'un **délai minimum de 100 ms** entre deux transitions.

### 4.3.5 Résultats et bilan technique

#### Fonctionnalités développées :

- ✓ **Mode MONITORING opérationnel** (transmission UART entre PC et capteur).
- ✓ **Détection du LIN et envoi d'un break LIN.**
- ✓ **Bascule automatique entre MONITORING et LIN.**

Même si la gestion **bidirectionnelle complète du LIN** n'a pas pu être finalisée, ce projet a permis de valider l'utilisation d'un **Arduino Due comme passerelle de communication** entre un PC et un capteur automobile.

## 5-Conclusion

### 5.1 Bilan du travail réalisé

Durant cette période en entreprise, j'ai eu l'occasion de travailler sur plusieurs projets techniques liés à l'Arduino et aux communications série, notamment en mode MONITORING et LIN. En suivant les consignes et les exemples donnés, j'ai pu avancer progressivement et résoudre les différents problèmes rencontrés. J'ai acquis des compétences dans la gestion des protocoles de communication, l'optimisation du code et l'utilisation des microcontrôleurs dans un environnement professionnel.

#### Projet 1 & Projet 2 : IHM Python et alimentation

Les premiers projets sur lesquels j'ai travaillé concernaient le développement d'une interface en Python pour le contrôle d'une alimentation. Ce travail m'a permis d'explorer l'utilisation de PyVISA pour communiquer avec des instruments électroniques via un port série. J'ai conçu une IHM permettant de configurer et de surveiller l'alimentation en temps réel, en assurant une communication fiable et optimisée.

Ce projet m'a appris à structurer un code Python orienté objet, à gérer les communications série de manière asynchrone et à garantir la stabilité de l'IHM face aux erreurs de transmission.

#### Projet final : Communication Arduino et protocole LIN

Le dernier projet, impliquant l'interface entre un capteur et un PC via l'Arduino Due, a été particulièrement enrichissant. Il m'a permis d'approfondir ma compréhension du protocole LIN, des interruptions série et de la manipulation des registres à bas niveau. Même si certaines parties n'ont pas pu être finalisées par manque de temps, l'essentiel des objectifs a été atteint, et une base solide a été mise en place pour les futurs développements.

### 5.2 Compétences acquises

En lien avec les **apprentissage critique (AC)** du BUT informatique, cette expérience m'a permis de développer de nombreuses compétences critiques :

- **AC1 - Concevoir et développer des solutions informatiques**
  - Programmation en **C++ pour microcontrôleurs** (Arduino).
  - Utilisation de **PyVISA** pour les communications en Python.
  - Implémentation et optimisation de **protocoles de communication (USART, LIN)**.
- **AC2 - Optimiser et déployer des applications**
  - Gestion des tampons série pour éviter les pertes de données.
  - Amélioration des performances des transmissions LIN.
  - Optimisation de code/refactoring.

- **AC3 - Administrer des systèmes informatiques et réseaux**
  - Configuration et exploitation d'un système embarqué.
  - Communication entre différents périphériques.
- **AC4 - Gérer un projet et travailler en équipe**
  - Interaction avec plusieurs collaborateurs (Jérôme, Éric, Damien, Théo).
  - Suivi des avancées et communication sur l'état du projet.
- **AC5 - Comprendre et s'intégrer dans un environnement professionnel**
  - Participation à des meetings d'entreprise.
  - Observation du fonctionnement inter-départements.
  - Adaptation aux contraintes et priorités industrielles.

### 5.3 Perspectives

Ce stage m'a apporté une expérience précieuse que je souhaite approfondir. Les connaissances acquises sur l'Arduino et les communications LIN pourront me servir dans de futurs projets académiques ou professionnels.

Dans le cadre de mes études en BUT Informatique, je compte capitaliser sur cette expérience pour aller plus loin dans :

- **L'intégration de systèmes embarqués** dans des projets réels.
- **L'automatisation et la robotique**, domaines que j'ai découverts lors de l'exposition interne de l'entreprise.

Par ailleurs, cette expérience m'a permis de comprendre l'importance des **interactions humaines** en entreprise. Je souhaite améliorer encore ma communication et mon travail en équipe pour de futurs projets collaboratifs.

### 5.4 Bilan de l'expérience en entreprise

Cette immersion dans le monde professionnel a été une expérience très enrichissante. J'ai eu l'opportunité d'observer le fonctionnement global d'une entreprise industrielle, en voyant comment les différents départements collaborent pour mener à bien des projets.

- J'ai assisté à des **meetings internes**, notamment sur les augmentations et primes, ainsi qu'à des élections d'entreprise.
- J'ai découvert **les différents bâtiments du site** (Ecar, E-nov, E-tech), chacun ayant ses propres spécialités.
- J'ai assisté à **une exposition interne** mettant en avant divers projets (robotique, mécanique, lumière, etc.).
- J'ai pu rencontrer des **collaborateurs** comme Jérôme, Damien, Théo et Éric, qui ont pris le temps de m'expliquer leur travail et de partager leur expérience.
- J'ai appris à **gérer les priorités** en anticipant les besoins futurs, notamment en prévoyant des mises à jour pour de nouvelles fonctionnalités.

Cette expérience m'a conforté dans mon choix de carrière et m'a donné une vision plus concrète du milieu professionnel.

## 5.5 Bilan de l'expérience personnelle

Sur le plan personnel, cette période a été tout aussi formatrice. J'ai beaucoup appris sur moi-même, notamment en termes de communication et de travail en équipe.

- **Sociabilisation** : J'ai appris à aller vers les autres, échanger avec différents collaborateurs et m'intégrer dans une dynamique de groupe.
- **Autonomie** : J'ai su gérer des tâches complexes seul, tout en demandant de l'aide lorsque nécessaire.
- **Montage et câblage** : J'ai été responsable du montage et de la configuration de l'Arduino sur le premier projet. Donc j'ai appris à faire de bonnes soudures, les schémas électroniques et d'autres.
- **Programmation** : J'ai développé des compétences en **C++ embarqué**, en **Python (PyVISA)** et en **protocoles de communication**.
- **Compréhension des systèmes embarqués** : Le dernier projet avec **LIN et USART** a été un vrai défi mais m'a permis de mieux comprendre le fonctionnement à bas niveau des microcontrôleurs.

Ce stage m'a énormément apporté, tant sur le plan technique que personnel. Il a renforcé mon intérêt pour l'informatique et m'a donné envie de poursuivre dans ce domaine, même si ça ne sera sûrement pas dans les réseaux ou la programmation à bas niveau. J'en ressors avec une meilleure vision des attentes du monde professionnel et une motivation pour la suite de mon parcours académique et professionnel.

### Signature :

Vue par M Godia le 26/03/2024

## 6 - Annexes :

### 6.1 Glossaire

#### A

- **AC (Apprentissages Critiques)** : Compétences spécifiques définies dans le cadre du BUT Informatique, évaluant les capacités de l'étudiant à concevoir, développer et administrer des systèmes informatiques.
- **Arduino** : Plateforme de développement open-source comprenant des cartes électroniques et un environnement de programmation destiné aux systèmes embarqués.

#### C

- **C++** : Langage de programmation orienté objet utilisé pour le développement des microcontrôleurs Arduino.
- **Communication Série** : Mode de transmission de données entre composants électroniques via des protocoles comme USART, LIN ou USB.

#### I

- **IHM (Interface Homme-Machine)** : Interface logicielle permettant l'interaction entre l'utilisateur et une machine.
- **Interruptions Série** : Mécanisme matériel permettant d'interrompre un programme en cours d'exécution lorsqu'une donnée est reçue sur une interface série.

#### L

- **LIN (Local Interconnect Network)** : Protocole de communication série utilisé principalement dans l'industrie automobile pour relier des capteurs et actionneurs à un microcontrôleur maître.

#### M

- **Microcontrôleur** : Circuit intégré programmable servant de cœur à des systèmes embarqués comme les cartes Arduino.
- **Monitoring** : Suivi en temps réel des performances et de l'état d'un système.

#### O

- **Objet (Programmation Orientée Objet - POO)** : Paradigme de programmation structurant le code sous forme d'objets réutilisables contenant des données et des méthodes.

#### P

- **PyVISA** : Bibliothèque Python permettant la communication avec des instruments électroniques via des interfaces série, USB ou Ethernet.
- **Protocole** : Ensemble de règles régissant la communication entre appareils électroniques ou logiciels.

## R

- **Registre (Microcontrôleur)** : Zone mémoire interne à un microcontrôleur utilisée pour configurer son comportement et stocker des valeurs temporaires.

## S

- **Système Embarqué** : Système informatique spécialisé, souvent intégré dans un appareil électronique et conçu pour une tâche spécifique.

## U

- **USART (Universal Synchronous/Asynchronous Receiver-Transmitter)** : Protocole de communication série permettant d'envoyer et de recevoir des données entre microcontrôleurs et périphériques.

## 6.2 Index des figures

### Index des figures

Figure 1: Utilisation de PyQt6 pour les champs de saisie.....	17
Figure 2: Validation uniquement float pour les valeur des volts et ampere.....	18
Figure 3: Envoie du switch depuis l'arduino.....	18
Figure 4: Reception du message de l'arduino via python.....	18
Figure 5: Montage Arduino du Projet 1.....	19
Figure 6: Détection de l'alimentation en USB.....	20
Figure 7: Fonction du changement de voltage de l'alimentation.....	20
Figure 8: Fichier de log projet 1.....	21
Figure 9: Validation des valeurs saisie pour UB et UB_Min.....	24
Figure 10: Première partie de l'ihm du projet 2.....	24
Figure 11: Start du programme du projet 2 avec les checkboxes.....	24
Figure 12: Code de la gestion des UB et UB_Min.....	25
Figure 13: Code du bouton stop clognotant.....	26
Figure 14: Management du déclencheur du blinking.....	26
Figure 15: initialisation des broches arduino pour le lcd.....	29
Figure 16: code d'affichage sur l'écran lcd.....	29
Figure 17: initialisation du lcd dans le setup de l'arduino.....	30
Figure 18: Code de gestion du mode monitoring.....	31
Figure 19: Code de détection du LIN beark.....	32
Figure 20: code du changement vers le LIN.....	32
Figure 21: Génération du break LIN.....	33
Figure 22: Fonction loop de l'Arduino.....	34