

Travail de Bachelor 2021

Logiciel de contrôle d'un robot de test de modules électroniques

Étudiant : Sébastien Deriaz

Travail proposé par : Gregory Hänggi

Betech SA

16, Rue du Four

1055 Froidenville

Enseignant responsable : Pierre Favrat

Année académique : 2020-2021

Yverdon-les-Bains, le 28 juillet 2021

Département Technologies industrielles (TIN)
Filière Génie électrique
Orientation Electronique embarquée et mécatronique
Étudiant Sébastien Deriaz
Enseignant responsable Pierre Favrat

Travail de Bachelor 2020-2021

Logiciel de contrôle d'un robot de test de modules électroniques

Betech SA

Résumé publiable

L'objectif de ce travail est la mise en œuvre d'un système permettant l'automatisation d'un banc de mesure de la société Betech SA. Ce banc, constitué d'un robot de manipulation et d'un ensemble d'appareils de laboratoire, permet la vérification et la caractérisation de modules radios au travers de différents tests : puissance d'émission, qualité de la réception, fréquence de transmission, communication UART, entre autres.

L'approche choisie consiste en la création d'une plateforme de tests et de mesures, polyvalente et réutilisable. Cette plateforme permet à l'utilisateur de décrire, au travers d'un script simple, toutes les opérations d'un banc de test : configuration des appareils, récupération de mesures, envoi de commandes, etc... Les opérations à effectuer sont traduites en instructions dans un script Python. Une interface graphique écrite en C++ assure une connexion bidirectionnelle entre le script et les appareils. L'interface fournit également des outils de test d'appareils, de diagnostic et de visualisation de données.

L'utilisateur peut réaliser le code Python dans un script (fichier .py) ou dans un notebook Jupyter (fichier .ipynb). L'utilisation des notebooks rend possible la documentation du banc de test grâce à des cellules markdown, ainsi que l'utilisation d'outils dynamiques de développement (autocomplete, recherche de fonctions disponibles pour chaque appareil, etc...). Des drivers peuvent être créés pour implémenter les commandes et le fonctionnement spécifique de chaque appareil. Ils permettent de simplifier l'écriture du code, tout en améliorant sa lisibilité. L'écriture de code haut niveau permet une description efficace des actions effectuées sur chaque appareil. À condition que le code Python se trouve sur la machine sur laquelle l'interface graphique a été démarrée, l'utilisateur est libre de choisir la méthode d'exécution.

Les appareils connectés via Ethernet, USB, GPIB ou série sont compatibles, à condition que leur protocole de communication soit standard et ouvert. Tous les appareils de laboratoire qui intègrent une méthode de communication, les machines avec commandes textuelles, ainsi que les périphériques sur bus de terrain sont compatibles (par exemple : oscilloscope, multimètre, analyseur de spectre, module d'entrées / sorties (ModBus TCP), machine CNC (GCode), onduleur pour production photovoltaïque (ModBus TCP) ou carte Arduino (UART), etc....). La configuration de chaque appareil est réalisée au travers de l'interface graphique. Chaque appareil est ensuite référencé dans le code Python à l'aide d'un alias.

Les données (mesures, valeurs de commandes, etc...) sont traitées dans le script à l'aide des librairies de Python

(NumPy, Pandas, Matplotlib, SciPy, etc...). Cette approche permet de tirer parti des capacités de Python pour le traitement de données. Le stockage s'effectue avec les librairies mentionnées précédemment, afin de créer des fichiers réutilisables (.csv, .dat, .bmp, etc...). Un système a également été mis en place pour enregistrer spécifiquement les données qui dépendent du temps (date et heure avec précision à la seconde). L'outil Grafana a été utilisé afin de visualiser les données en fonction du temps, sur une interface web accessible depuis un ordinateur distant. La combinaison du système de stockage et de visualisation est particulièrement adaptée à des tâches de monitoring.

Un gestionnaire de projet est mis en place afin d'organiser les scripts et les données de chaque banc de test. Un projet, stocké sous forme de dossier, peut être transmis à d'autres utilisateurs.

La travail fourni respecte le cahier des charges. Tous les objectifs, primaires comme secondaires, ont pu être réalisés. Un banc de test a été écrit et executé avec succès sur un circuit radio.

L'orientation polyvalente du travail permet de l'utiliser pour d'autres projets dans le cadre de l'école (laboratoires avec instruments de mesures, tests sur des bus de terrain, etc...)

Étudiant	Date et lieu :	Signature
Sébastien Deriaz	_____	_____
Enseignant responsable	Date et lieu :	Signature
Pierre Favrat	_____	_____
Betech SA	Date et lieu :	Signature
Gregory Hänggi	_____	_____

Préambule

Ce travail de Bachelor (ci-après TB) est réalisé en fin de cursus d'études, en vue de l'obtention du titre de Bachelor of Science HES-SO en Ingénierie.

En tant que travail académique, son contenu, sans préjuger de sa valeur, n'engage ni la responsabilité de l'auteur, ni celles du jury du travail de Bachelor et de l'Ecole.

Toute utilisation, même partielle, de ce TB doit être faite dans le respect du droit d'auteur.

HEIG-VD

Le Chef du Département

Yverdon-les-Bains, le 28 juillet 2021

Authentification

Sébastien Deriaz atteste par la présente avoir réalisé ce travail et n'avoir utilisé aucune autre source que celles expressément mentionnées.

Yverdon-les-Bains, le 28 juillet 2021

Sébastien Deriaz

Table des matières

1	Introduction	8
1.1	Description du système	8
1.1.1	Robot	9
1.1.2	Appareils	11
1.2	Approche	12
2	Analyse	13
2.1	Lecture / Écriture	13
2.2	Communication avec les appareils	13
2.3	Scripts	17
2.3.1	Choix du langage	18
2.3.2	Fichiers de correspondance (drivers)	19
2.4	Environnement de programmation	19
2.4.1	Remarque	20
2.5	Données	20
2.5.1	Données temporelles et événements	20
2.5.2	Types de données de travail	21
2.5.3	Outil Grafana (objectif secondaire)	22
2.5.4	Base de données	23
2.6	Notebooks Jupyter	23
3	Réalisation	24
3.1	Approche	24
3.1.1	Architecture	25
3.2	Interaction avec l'utilisateur	25
3.3	Programme C++	26
3.4	Gestion des appareils	27
3.4.1	Communication avec les appareils	27
3.4.2	Ethernet	28
3.4.3	USB	28
3.4.4	GPIB	28
3.4.5	Série	28
3.5	Interaction avec les scripts	29
3.6	Gestion des requêtes	29
3.7	Lancement des scripts	29
3.8	Données	30
3.8.1	Blocs	30
3.8.2	Affichage des données	31
3.9	Projets	31
3.10	Interface graphique	32
3.10.1	Organisation	32
3.11	Package Python	38
3.11.1	Drivers	38
3.12	Licences	39

3.12.1 Choix de licence pour le travail	39
4 Application	40
4.1 Module RM1S2	40
4.2 Protocole	41
4.2.1 Initialisation	41
4.2.2 Déplacement du robot	41
4.2.3 Programmation	42
4.2.4 Mesure de la puissance d'émission	42
4.2.5 Mesure du Bit Error Rate	43
4.2.6 Test de communication entre deux modules	43
4.2.7 Script	44
4.3 Mesures	49
5 Conclusion	51
5.1 Cahier des charges	51
5.1.1 Objectifs primaires	51
5.1.2 Objectifs secondaires	51
5.2 Plateforme	51
5.2.1 Appareils	51
5.2.2 Drivers	52
5.2.3 Scripts	52
5.3 Améliorations possibles	52
5.3.1 Interface graphique	52
5.3.2 Données	52
5.3.3 Système	52
5.3.4 Appareils et communication	52
5.3.5 Ajouts	52
6 Bibliographie	53
Table des figures	55
Liste des tableaux	56
A Cahier des charges	57
B Manuel d'utilisation	68
C Planning (24.07.2021)	84
D Licence VISA Shared Components	86

Chapitre 1

Introduction

L'objectif initial du travail est la mise en œuvre d'un système de banc de test pour la société Betech SA[32].

1.1 Description du système

Le banc de test est composé de deux parties superposées :

1. Robot "gantry" : Machine de type CNC¹ qui permet de déplacer une tête de mesure sur des circuits
2. Instruments de mesures pour la caractérisation du circuit (alimentation, appareils RF, générateur etc...)

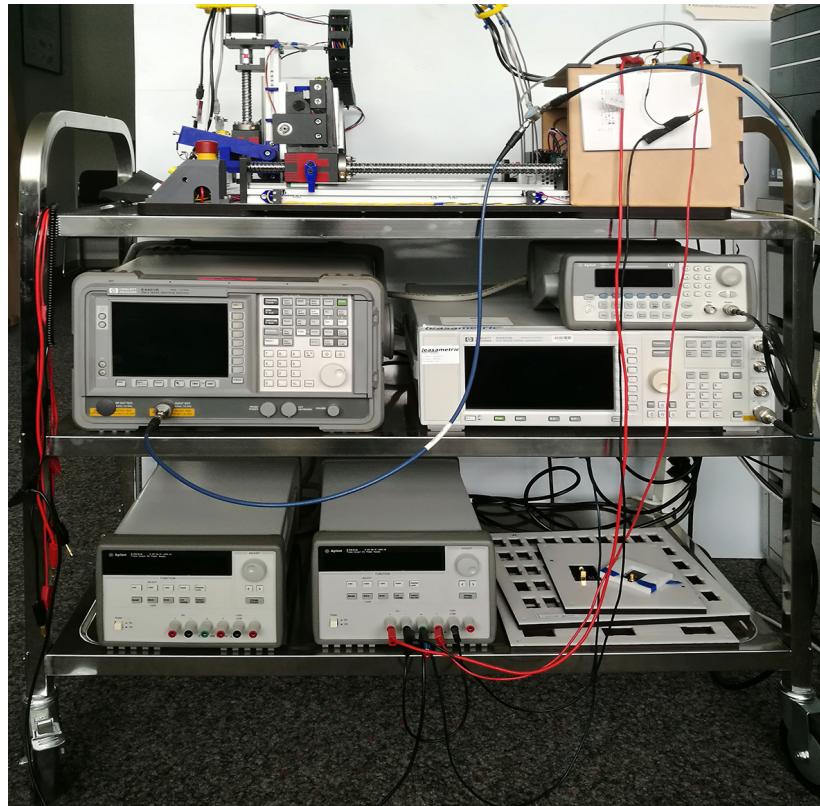


FIGURE 1.1 – Instruments de mesure

1. Computer Numerical Control

1.1.1 Robot

La tête de mesure est constituée d'un PCB auquel sont fixés des "pogo-pins" qui permettent la connexion avec le circuit testé

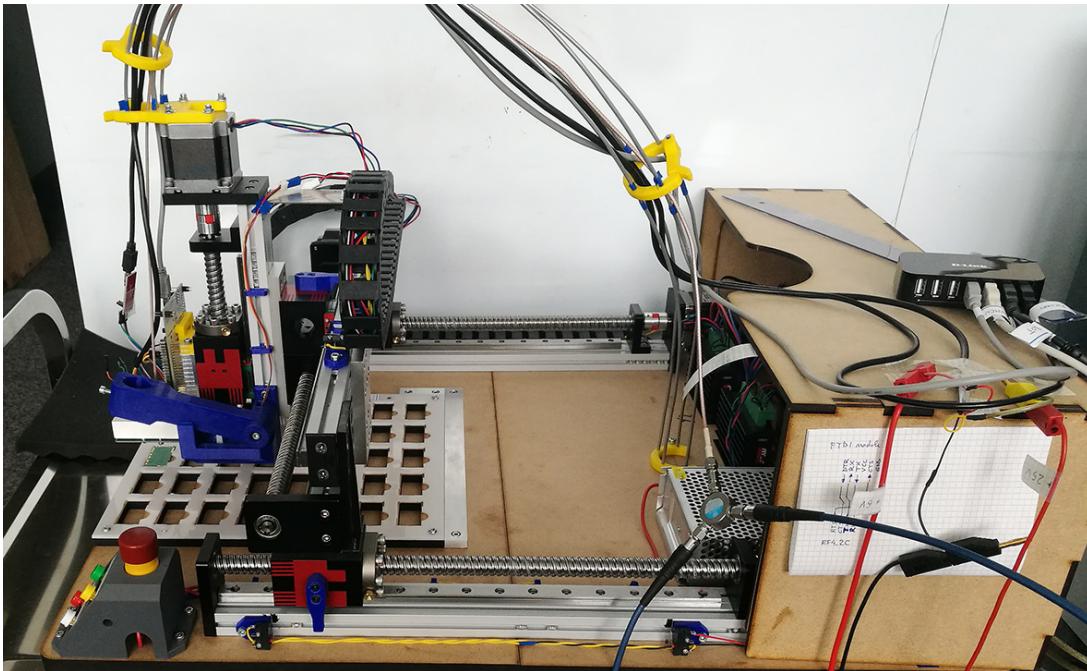


FIGURE 1.2 – Robot

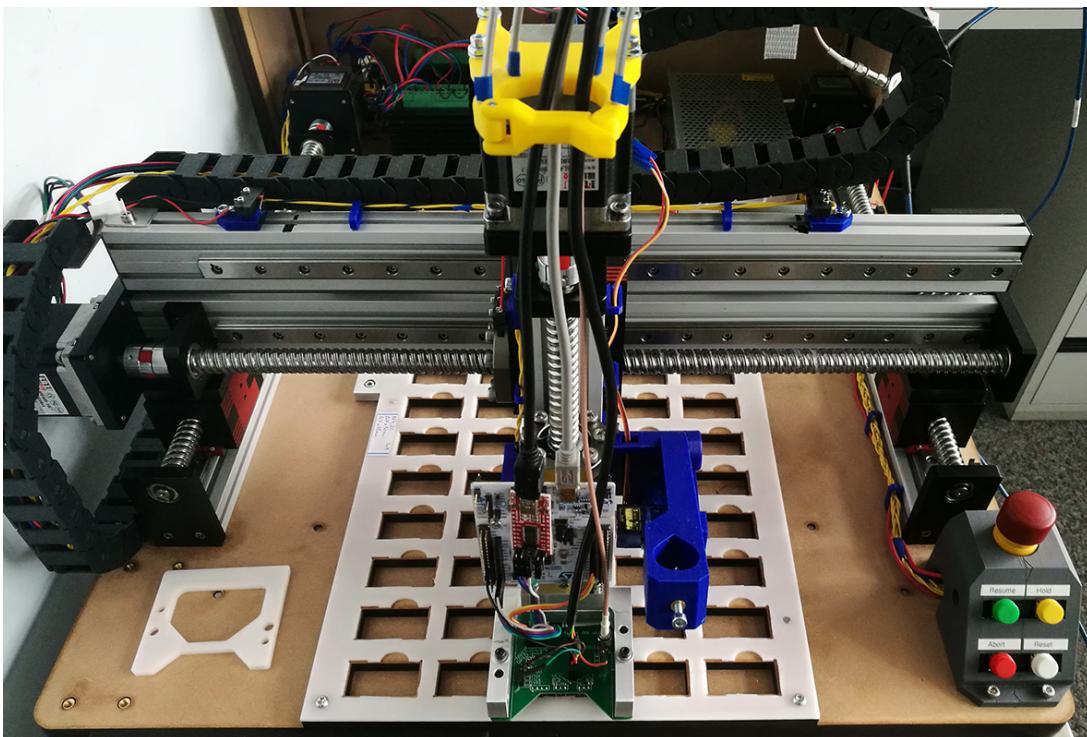


FIGURE 1.3 – Tête de mesure

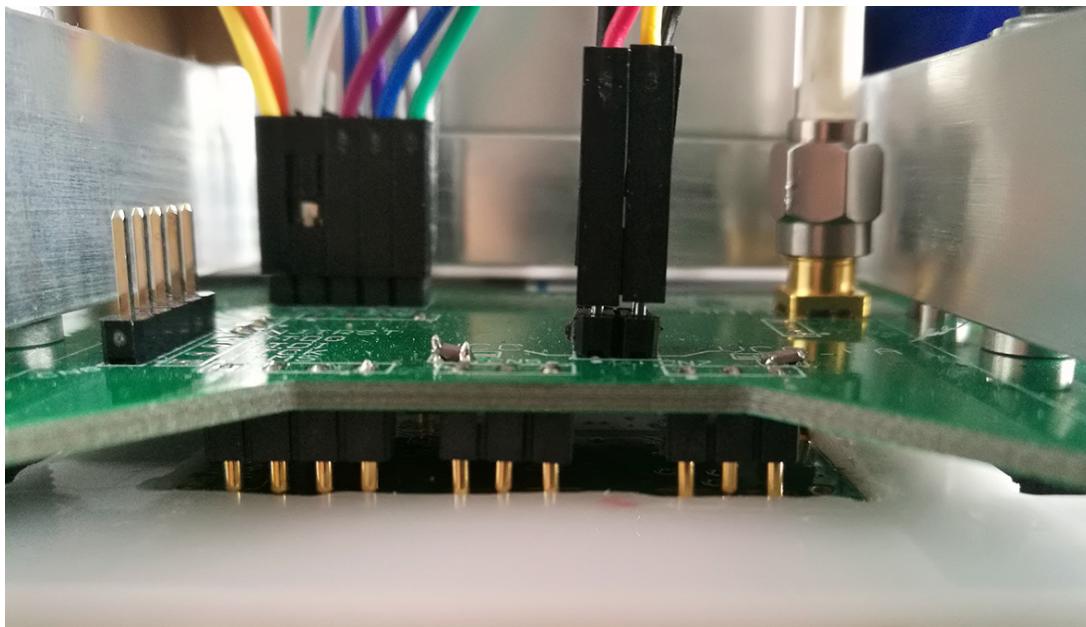


FIGURE 1.4 – Tête de mesure placée sur un circuit

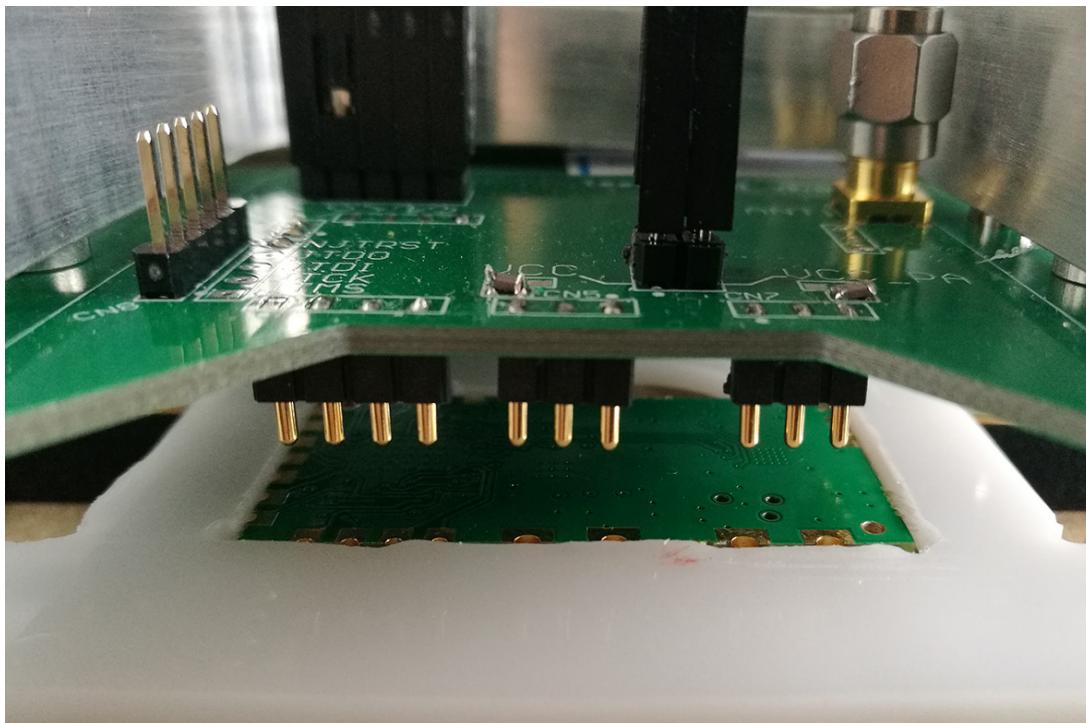


FIGURE 1.5 – Circuit et "pogo-pins"

1.1.2 Appareils

Appareil	Connexion	Protocole	Modèle ²
Robot	USB	Série	FUYU FSL40XYZ-L
Interface avec le module radio	USB	Série	FT232 (USB-Serial)
Contrôle du module radio	USB	Série	STM32Nucleo
Alimentation	GPIB	SCPI	Agilent E3631A
Analyseur de spectre	GPIB	SCPI	HP E4401B
Générateur RF	GPIB	SCPI	HP E4421B
Générateur de signal	GPIB	SCPI	Agilent 33220A
Interface GPIB	USB	-	Agilent 82357B

TABLE 1.1 – Liste des appareils

Les appareils série sont reliés en USB grâce à des interfaces USB-Série (convertisseur ou intégré au système). Les appareils GPIB sont reliés ensemble via des câbles GPIB puis au PC avec une interface USB-GPIB.

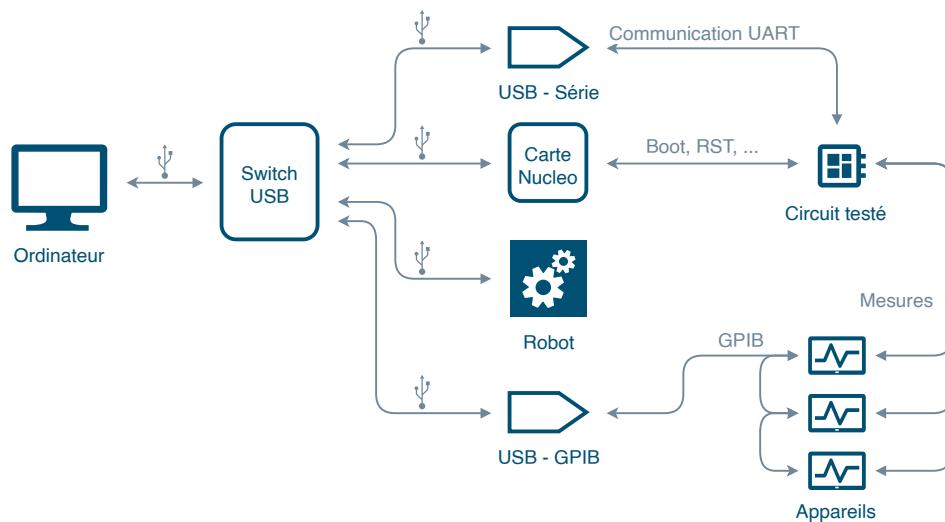


FIGURE 1.6 – Schéma de connexion des appareils

Le robot a été réalisé par Marc-Antoine Maillard en 2020. La documentation[20] est disponible en annexe.

2. Si disponible

1.2 Approche

L'approche choisie est la réalisation d'un logiciel de contrôle universel capable de communiquer avec différents types d'appareils :

- Équipement de laboratoire
- Appareils avec communication standardisée (Série, Modbus TCP)

Cette approche permet de réaliser un système polyvalent qui peut être réutilisé pour d'autres projets comme :

- Bancs de tests électroniques (laboratoires par exemple)
- Tests rapides sur des bus de communications (Modbus TCP, UART)

Toutes les fonctions demandées dans le banc de test de la société Betech doivent pouvoir être réalisées.

Chapitre 2

Analyse

2.1 Lecture / Écriture

La très grande majorité des interactions avec un périphérique peut se résumer à deux opérations :

- Lecture : Périphérique → hôte
- Écriture : Hôte → périphérique

Par exemple, les appareils de laboratoire qui utilisent le protocole SCPI[30] sont entièrement contrôlables par des commandes ASCII (écriture). Certaines commandes, lorsqu'elles sont envoyées à l'appareil, génèrent des données en retour qui peuvent être récupérées (lecture).

L'écriture se fait toujours sur demande de l'utilisateur. La lecture peut s'effectuer de deux manières :

- Synchrone : (LXI ou GPIB par exemple). Une demande (spécifique au protocole de communication) doit être faite afin de récupérer les données
- Asynchrone : Les données sont transmises par l'appareil lorsqu'elles sont prêtes (Port série par exemple). Dans ce cas, un buffer doit stocker les données avant qu'elles soient demandées par l'utilisateur

L'utilisateur peut demander une lecture et ignorer le résultat pour effacer le buffer (s'il existe). Cette méthode rend entièrement transparente le type de communication avec l'appareil.

2.2 Communication avec les appareils

L'architecture de communication avec les appareils est déjà existante et sous-utilisée la plupart du temps. La totalité des appareils de laboratoire (oscilloscopes, alimentations, etc...) possède une forme de communication qui permet le paramétrage et la mesure.

Le maximum de méthodes de communication devront être supportées afin de maximiser la flexibilité de la plateforme. Si un appareil possède plusieurs méthodes de communication, la plus performante (ou la plus pratique) devra être utilisée. Le choix sera laissé à l'utilisateur

La table 2.1 donne des exemples¹ de types d'appareils et de leurs protocoles de communication

1. Ces exemples sont tirés du matériel à disposition lors du travail, beaucoup d'autres combinaisons existent

Appareil	Connexion	Transport	Protocole
Oscilloscope, Multimètre de table, Alimentation de laboratoire, Générateur de fonction	Ethernet	TCP/UDP ²	SCPI
	Ethernet	VXI	SCPI
	USB	USBTMC	SCPI
	Câble GPIB	GPIB	SCPI
	Câble DB9	RS-232	SCPI
Module d'entrées-sorties modbus TCP ³	Ethernet	TCP	Modbus (TCP)
Imprimante 3D / robot (CNC)	USB (Adaptateur USB-Série)	UART	GCode
Onduleur pour installation photovoltaïque	Ethernet	TCP	Modbus (TCP)
Appareil communicant en UART (arduino, module RF, etc...)	USB (Adaptateur USB-Série)	UART	libre ⁴
Multimètre à main ⁵	USB (Adaptateur USB-Série)	USB	spécifique ⁶

TABLE 2.1 – Exemples de combinaisons protocole / méthode de communication

Les 4 méthodes de connexion suivantes devront être gérées :

1. Ethernet
2. USB
3. GPIB
4. Série

L'objectif est d'obtenir un appareil contrôlable (lecture et écriture) dissocié de sa méthode de communication. L'appareil sera configuré une fois, puis référencé par un alias (par exemple "mult" pour un multimètre).

Ethernet

Un appareil Ethernet doit être connecté sur le même réseau que l'ordinateur qui le contrôle. Seule l'adresse (IPv4 ou IPv6) est nécessaire pour l'identifier. Le port peut être spécifié en fonction du protocole utilisé. La communication s'effectue en UDP ou en TCP, en fonction du protocole et du choix de l'utilisateur.

La commande "ping" (protocole ICMP) peut être utilisée pour tester la présence de l'appareil.

Dans le cas des appareils LXI[12], un serveur HTTP est automatiquement créé. Il est donc possible d'envoyer une requête HTTP et de tester un retour, ce qui permet de supposer que l'appareil est compatible LXI.

2. La plupart du temps TCP car UDP n'est pas fiable

3. Par exemple ADAM-6266

4. Le protocole est défini par l'utilisateur ou le fournisseur du système

5. Par exemple UNI-T 61E

6. Les données sont envoyées en continu à l'hôte

USB

Chaque appareil USB doit être relié à l'ordinateur soit directement, soit par l'intermédiaire d'un hub. Les appareils de laboratoire compatibles USB utilisant tous le protocole USBTMC, seule une librairie spécifique pour USBTMC est nécessaire (au lieu de gérer la couche USB complète).

La librairie VISA[11] de Keysight permet de simplifier la connexion aux appareils, ainsi que de fournir une liste des appareils disponibles.

Les appareils USB sont référencés grâce à une adresse standardisée de la forme

```
USB0::0xF4EC::0xEE3A::SDS00003132994::INSTR
```

Pour un appareil de numéro de série SDS00003132994 connecté sur le hub usb 0 avec un VID de 0xF4EC et un PID de 0xEE3A[5]

GPIB

Les appareils GPIB sont reliés par des câbles du même nom (voir figure 2.1)

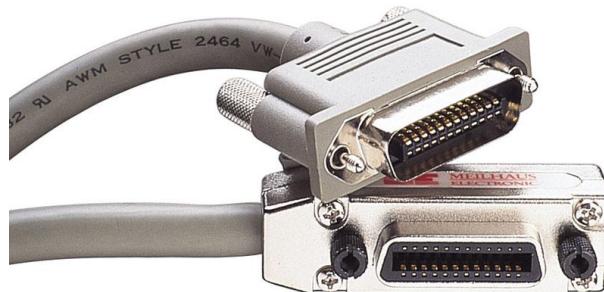


FIGURE 2.1 – Connecteurs GPIB⁷

Ces câbles permettent de relier plusieurs appareils en *daisy-chain*⁸. Un adaptateur USB-GPIB est nécessaire pour relier les appareils à l'ordinateur. L'adaptateur utilisé pour ce travail sera le Keysight 82357B⁹.

Comme pour les appareils USB, la librairie VISA, qui met en place la couche de communication complète avec l'adaptateur USB, sera utilisée. Les appareils GPIB sont référencés avec un descripteur similaire à l'USB, mais qui fait intervenir l'adresse GPIB¹⁰, de la forme :

```
GPIB0::23::INSTR
```

Dans ce cas, l'appareil est connecté sur l'interface GPIB0 avec l'adresse 23.

Plusieurs interfaces GPIB peuvent être présentes sur un même ordinateur. Afin d'éviter une confusion, une liste des appareils disponibles devra être présentée à l'utilisateur.

7. source : <https://www.meilhaus.de/cosmoshop/default/pix/a/n/1274089177-8975.2.jpg>

8. [https://en.wikipedia.org/wiki/Daisy_chain_\(electrical_engineering\)](https://en.wikipedia.org/wiki/Daisy_chain_(electrical_engineering))

9. <https://www.keysight.com/ch/de/product/82357B/usb-gpib-interface-high-speed-usb-2-0.html>

10. Adresse unique dans un ensemble d'appareils reliés comprise entre 1 et 31

Série

Chaque appareil série doit être relié directement à l'ordinateur par l'intermédiaire d'un adaptateur Série-USB (voir figure 2.2)



FIGURE 2.2 – Adaptateur USB RS-232¹¹

La connexion à chaque appareil se fait avec son numéro de port (par exemple COM4).

Il est également possible de connecter des interfaces séries qui n'utilisent pas le connecteur RS-232, notamment des interfaces FTDI (USB vers UART) (voir figure 2.3)

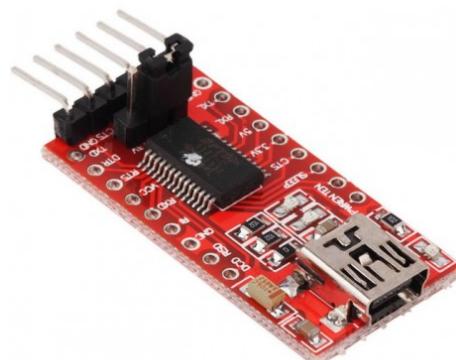


FIGURE 2.3 – Adaptateur USB - UART (FTDI)

11. source : https://images-na.ssl-images-amazon.com/images/I/51%2Bi6kHKOZL._AC_SY355_.jpg

2.3 Scripts

Il existe trois méthodes de description de banc de test :

1. Application dédiée
2. Interface graphique en "blocs"
3. Script

Une application dédiée (comme un logiciel spécifique à un oscilloscope) est extrêmement simple à prendre en main pour l'utilisateur. En revanche, l'application n'est pas évolutive et l'intervention de l'utilisateur est nécessaire pour effectuer les opérations (appui sur des boutons).

L'interface graphique en "blocs" (comme LabView[19]) permet une plus grande flexibilité, car les blocs sont universels mais des opérations de traitement de données sont plus difficiles. L'environnement LabVIEW est fermé, ce qui rend compliqué la communication avec d'autres appareils dont les drivers n'existent pas.

L'option des scripts qui sera retenue pour ce projet permet une meilleure flexibilité, à condition qu'une forme de documentation aide l'utilisateur à réaliser le script. Les scripts sont également très adaptés pour l'automatisation : il est possible d'appeler un script à partir d'un autre, ce qui ouvre les portes à la création de bancs de tests modulaires.

2.3.1 Choix du langage

Le choix du langage de script est critique et dicte le reste du développement du projet. Les langages les plus communs sont :

- C/C++ : langage compilé très utilisé mais de bas niveau
- C# : langage compilé de haut niveau, mais principalement limité à Windows
- Python : langage interprété haut niveau cross-plateforme et très utilisé dans le département TIN¹²
- Javascript : langage interprété haut niveau cross-plateforme

Les langages interprétés permettent une exécution plus facile, mais des erreurs sont parfois détectées lorsque la ligne est exécutée. Ceci peut être problématique pour un banc de test qui fonctionne sur plusieurs heures.

La figure 2.4 montre une comparaison des différents langages

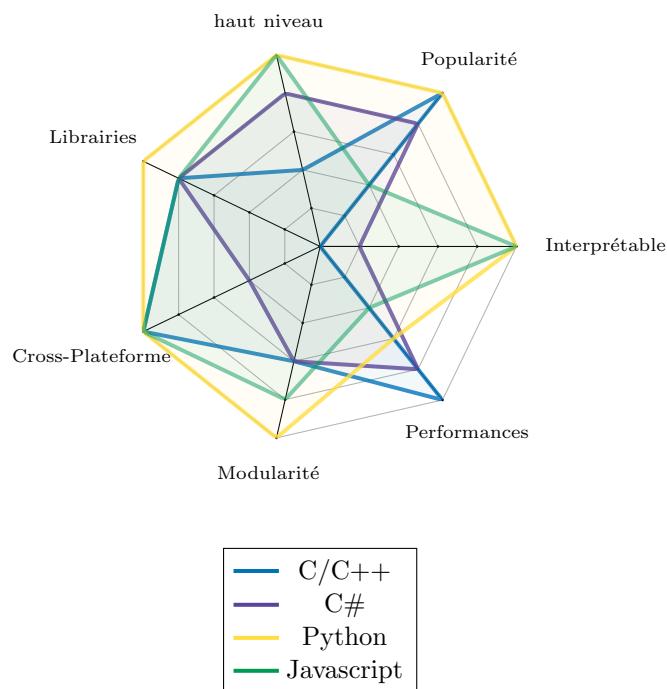


FIGURE 2.4 – Comparaison des langages

Le langage qui sera retenu est Python, car il allie une grande modularité avec des librairies performantes (NumPy[15], Pandas[25], Matplotlib[14]) qui sont utilisées dans le traitement de données.

L'utilisation de Python ouvre également la porte aux notebooks Jupyter[23], ce qui permet d'exécuter du code par étapes et d'effectuer des tests rapidement.

Les scripts peuvent être organisés en modules et sous-modules, afin d'encapsuler des fonctionnalités. Par exemple un module peut être réalisé pour effectuer un test en particulier sur un système. Ce module sera ensuite appelé plusieurs fois en fonction du contexte.

12. Département des Technologies industrielles

Exemple

Un script doit être écrit de la manière la plus simple possible. Chaque appareil est représenté par un objet et les méthodes liées permettent d'effectuer les actions spécifiques

```

1 # Exemple d'une initialisation d'un appareil
2 alim = device('alim_alias')
3
4 # Tension 10.5V sur le canal 1
5 alim.setDCVoltage(channel=1, voltage=10.5)
6
7 # Mesure du courant sur le canal 1
8 current = alim.getDCCurrent(channel=1)

```

FIGURE 2.5 – Exemple de script Python avec une alimentation

2.3.2 Fichiers de correspondance (drivers)

Un système de fichiers de correspondance est créé pour lier les fonctions *haut niveau*[13] avec les commandes et la syntaxe spécifique de chaque appareil. Dans le cas d'un multimètre Agilent 34401A, la commande pour effectuer une mesure de tension DC est MEAS:VOLT:DC?\n. Il est donc nécessaire de créer une fonction `measureDCVoltage()` qui envoie la commande correspondante à l'appareil, puis lit la réponse

MEAS:VOLT:DC?\n	→	<pre> 1 def measureDCVoltage(self): 2 self.device.write('MEAS:VOLT:DC?\n') 3 return float(self.device.read()) </pre>
-----------------	---	--

2.4 Environnement de programmation

Il est nécessaire de réaliser une interface graphique en parallèle de la gestion de la communication avec les appareils. Afin de limiter le nombre de programmes, les deux parties seront réalisées ensemble. Les options suivantes sont les plus courantes pour réaliser un programme avec une interface graphique :

- C# avec une interface WPF¹³ / WinForms dans Visual Studio
- C++ avec QML¹⁴ dans l'environnement Qt
- Python avec Tkinter¹⁵ avec PyCharm

Bien que le C# et le WPF soient plus adaptés pour la réalisation rapide d'une interface utilisateur, ils sont désavantageés par leur exclusivité Windows. La librairie Tkinter avec Python ne permet pas de réaliser facilement des interfaces graphiques complexes et, par conséquent, c'est l'option C++/QML qui sera retenue.

L'environnement Qt permet de réaliser des interfaces graphiques complexes et met à disposition des librairies performantes pour simplifier le code en C++ (gestion des ports série, des bases de données, des threads etc...). Le cross-plateforme est également encouragé, ce qui peut être un avantage pour la suite du projet (utilisation sur Raspberry Pi par exemple).

De nombreuses librairies sont disponibles en C ou C++ pour la connexion aux appareils notamment :

- QSerialPort[7] : port série
- VISA[11] : USBTMC et GPIB
- QAbstractSocket[6] : communication ethernet

Les librairies Qt seront utilisées dans tout le programme pour diverses fonctionnalités.

13. Langage de programmation adapté pour les interfaces graphiques

14. Équivalent de WPF dans l'environnement Qt

15. Librairie Python pour la création d'interfaces graphiques

2.4.1 Remarque

Il existe deux méthodes pour créer une interface graphique :

- Visuelle : "glisser-déposer" de blocs sur un canvas
- Programmée : programmation de l'interface avec un langage spécifique (comme WPF ou QML)

La deuxième méthode est utilisée, car elle est plus modulaire et permet de créer des interfaces graphiques plus complexes.

2.5 Données

Il est nécessaire de distinguer plusieurs types de données lors de l'exécution du programme

1. Données de transmission : commandes envoyées et reçues par le système aux appareils
 - Durée de vie très courte (le temps d'un envoi ou d'une réception)
 - Indication éventuelle à l'utilisateur des requêtes en cours
 - Aucun enregistrement
2. Données de statut : Information sur l'exécution du programme avec potentiellement un niveau d'information (info, warning, error etc...)
 - Affichage à l'utilisateur d'un indicateur de couleur en fonction du niveau
 - Faible volume
 - Enregistrement à l'arrêt du programme facultatif
3. Données temporelles "lentes" ou événements : statut d'un système à une date précise, progression du système sur des minutes/heures/jours, température ambiante, etc...
 - Stockage avec *timestamp*¹⁶
 - Grand nombre d'enregistrements, mais chacun est peu volumineux
 - Doivent être conservées après l'arrêt du programme
4. Données de travail : oscilloscopes, captures d'écran, tableaux de données etc...
 - Doivent pouvoir être réutilisées et/ou transmises.
 - Stockage dans des formats courants (valeurs numériques en csv, dat ou txt et images en png ou jpg)
 - Traitement dans Python
 - Peu d'éléments, mais chacun est potentiellement volumineux
 - Doivent être conservées après l'arrêt du programme

Les données de transmission sont stockées le temps que l'appareil soit disponible, puis sont transmises à l'appareil (écriture) ou au script Python (lecture).

Les données de mesure sont présentées à l'utilisateur sous forme de texte ou de tableaux NumPy[15]. L'utilisateur est libre de les stocker dans des fichiers spécifiques en utilisant les outils de Python (NumPy[15], Matplotlib[14], SciPy[16] etc...)

2.5.1 Données temporelles et événements

Les données temporelles (valeur à une date précise, température, progression, etc...), ainsi que les événements, doivent être stockés avec un *timestamp*[9], c'est à dire avec une indication du temps précis auquel l'événement est arrivé (date du jour avec précision à la seconde). Il existe deux possibilités :

- Option de base : laisser l'utilisateur stocker les valeurs temporelles comme des données de mesure. Il est responsable de la méthode de stockage et de la présentation des informations.
- Amélioration (objectif secondaire) : Stocker les valeurs dans une base de données ce qui permet un accès rapide par d'autres programmes (voir 2.5.3). Un *timestamp*[9] est ajouté automatiquement.

Les données de statut seront traitées de la même manière. Ceci permet d'y ajouter la notion de temps et le stockage peut s'effectuer dans la base de donnée (si l'objectif secondaire est réalisé).

16. Indicateur de temps absolu (date et heure)

2.5.2 Types de données de travail

Il existe 4 types principaux :

1. Textes
2. Valeurs numériques ponctuelles
3. Tableaux de valeurs (Oscillogrammes, séquence mesurée, etc...)
4. Images (Tableaux 2D ou 3x2D)

Textes

Le texte peut être sauvegardé dans des fichiers .csv ou .txt facilement avec Python. Si le texte correspond à un événement ponctuel, il est possible de le stocker dans un tableau avec une colonne pour la date précise. L'affichage peut se faire avec le système de stockage, ou directement dans l'interface.

Valeurs numériques

Comme pour le texte, les valeurs numériques ponctuelles peuvent être stockées dans des fichiers .csv, .txt, ou dans des tableaux avec date pour les lier à un temps. Le stockage dans des tableaux permet d'obtenir un graph si nécessaire.

Tableaux de valeurs

Comme les tableaux de valeurs peuvent être volumineux et nécessitent parfois des axes spécifiques (temps à la nanoseconde, fréquence, etc...), il est plus adapté de les traiter dans Python à l'aide de librairies comme NumPy[15], puis de stocker les données dans des fichiers .dat ou .csv

Images

Les images doivent être traitées dans Python, puis stockées dans des fichiers .bmp, .jpg ou .png. Cette action est notamment possible grâce à la librairie Matplotlib[14].

2.5.3 Outil Grafana (objectif secondaire)

Grafana[24] est un outil de visualisation. Il permet d'afficher des informations stockées dans des bases de données sous forme de "dashboards". Les méthodes de visualisations principales sont :

- Graph
- Bargraph
- Stat
- Table
- Gauge
- Log

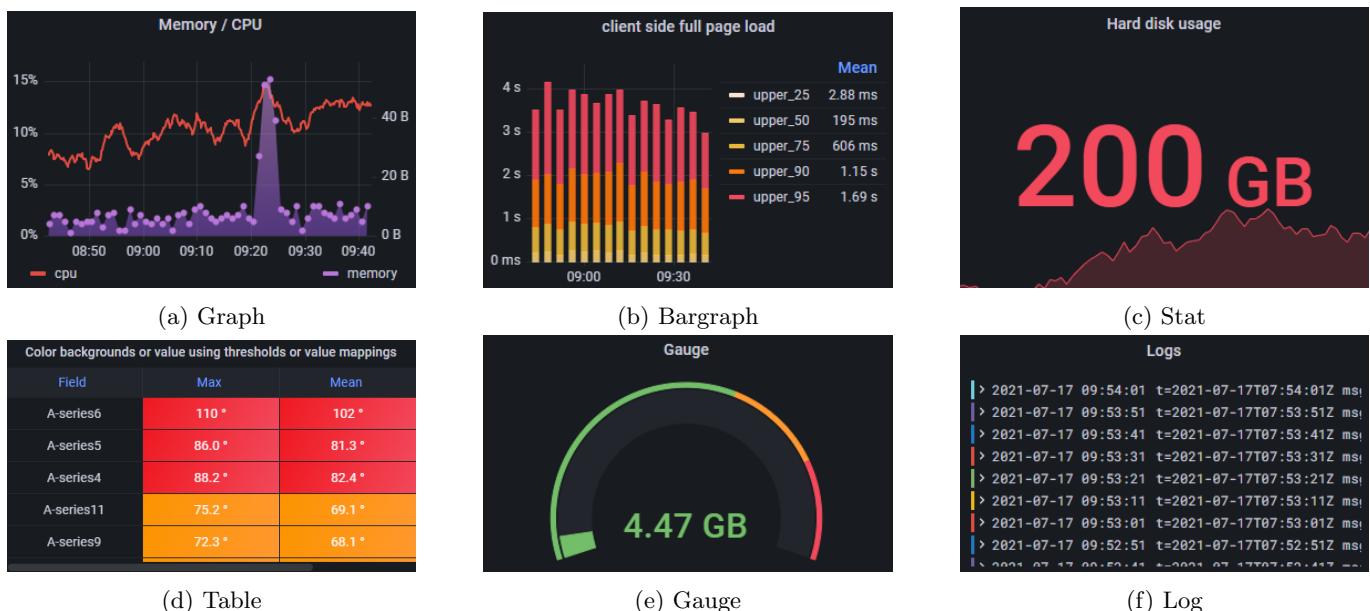


FIGURE 2.6 – Exemples de visualisations Grafana. Source : <https://play.grafana.org/>

Les visualisations graph, bargraph, stat et gauge sont utilisées pour des données numériques qui varient en fonction du temps. La table est utilisée pour afficher des données sur deux axes simultanément. Les logs permettent d'afficher des événements à une date précise. Ce dernier pourrait afficher les données status (voir 2.5).

À ce jour, Grafana n'est pas prévu pour lire des données provenant de fichiers texte (.csv, .dat, etc...) et la précision de temps est limitée à la seconde. Il n'est également pas possible de choisir un axe différent du temps pour les graphs (et autres visualisations qui dépendent du temps). Pour cette raison, Grafana serait utilisé uniquement pour les données temporelles lentes ou l'affichage sous forme de table (pas d'oscillosogrammes, courbes d'analyseur de spectre, etc...)

Accès réseau

Grafana fonctionne avec un serveur web (serveur sur l'ordinateur hôte). Il est possible¹⁷ de visualiser les données depuis un autre ordinateur sur le même réseau, voir en dehors du réseau. Ceci permettrait d'observer le fonctionnement du banc de test à distance, particulièrement lorsque l'exécution peut prendre plusieurs heures.

17. A condition que le port 3000 soit ouvert en sortie

2.5.4 Base de données

Pour stocker les données et permettre leur accès par Grafana, il est nécessaire d'utiliser une base de données[2] comme MySQL[3] ou SQLite[4].

Une base de données permet de stocker de l'information dans des "tables" (similaires à une feuille dans un classeur excel) et de les relire par la suite. Toutes les interactions se font avec des commandes textuelles traitée par le moteur de la base de données. Dans le cas de MySQL il s'agit d'un serveur qu'il faut démarrer. SQLite ne nécessite pas de serveur et une librairie C++ ou Python se charge de traiter les requêtes.

Ce système de requêtes textuelles est exploité par Grafana qui va effectuer des requêtes de lectures à intervalles réguliers (typiquement 5 secondes) sur la base de données, pour afficher les informations.

Afin d'éviter de multiplier le nombre de programmes à utiliser pour ce projet, l'idéal est d'utiliser SQLite, car aucun serveur n'est nécessaire. La base de données est contenue dans un fichier .db exclusivement et l'accès se fait au travers d'une librairie C++ ou Python. L'inconvénient de cette méthode est qu'un seul client peut écrire sur la base à la fois (verrouillage). La lecture, en revanche, est possible par plusieurs clients. En pratique, cela signifie que si un script Python ouvre la base de données, les autres programmes (interface graphique, Grafana) peuvent uniquement effectuer des lectures sur la base de données.

2.6 Notebooks Jupyter

Les scripts Python (fichiers .py) sont faciles à utiliser et il est possible de créer des systèmes modulaires (scripts qui appellent d'autres scripts).

Il existe toutefois une autre méthode pour exécuter du code Python : les notebooks Jupyter[23]. Ils permettent d'exécuter manuellement des morceaux de code (appelés cellules) les uns après les autres. Cette architecture permet d'effectuer des tests bien plus rapidement.

Il est également possible d'intercaler du texte ou des images entre les morceaux de code, ce qui permet de documenter le script efficacement.

L'auto-complétion de code dans un notebook permet de détecter les fonctions disponibles pour chaque appareil.

Chapitre 3

Réalisation

3.1 Approche

L'approche choisie est la création d'une librairie Python communiquant avec un programme en C++. La librairie Python permet à l'utilisateur de lancer des scripts haut niveau. Le programme C++ gère la communication avec les appareils et fournit une interface graphique pour la configuration des appareils, le diagnostic et l'affichage des données. Un système d'abstraction est créé pour transformer les commandes spécifiques à chaque appareil en méthodes Python. L'utilisateur peut ainsi créer des scripts de manière simple et intuitive. Par exemple :

- Déplacer la machine aux coordonnées X=10.5, Y=19.5
- Appliquer une tension de 5V via l'alimentation
- Envoyer la commande "START" via l'interface UART
- Obtenir la fréquence d'amplitude maximal mesurée par l'analyseur de spectre
- etc...

La communication entre la librairie Python et le programme C++ se fait à travers un canal TCP.

Des options de diagnostic et de statut sont ajoutées pour aider l'utilisateur à identifier les éventuels problèmes (erreur de connexion à un appareil, commandes erronées, etc...).

Le projet sera séparé en trois parties :

1. Communication avec les appareils
2. Gestion des données
3. Interaction avec l'utilisateur (interface graphique et scripts)

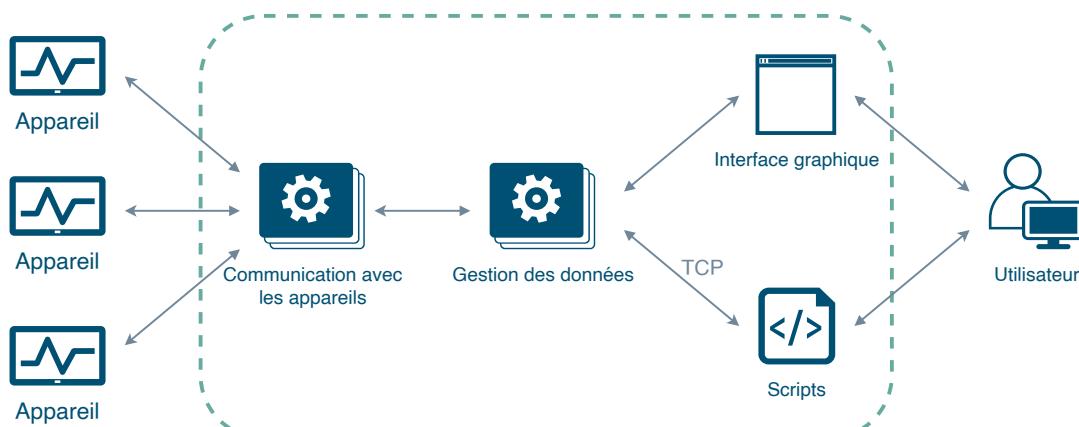


FIGURE 3.1 – Structure du projet

La zone entourée indique la partie à réaliser dans le projet.

3.1.1 Architecture

La figure 3.2 montre le flux des données dans le système avec l'objectif secondaire de stockage dans une base de donnée.

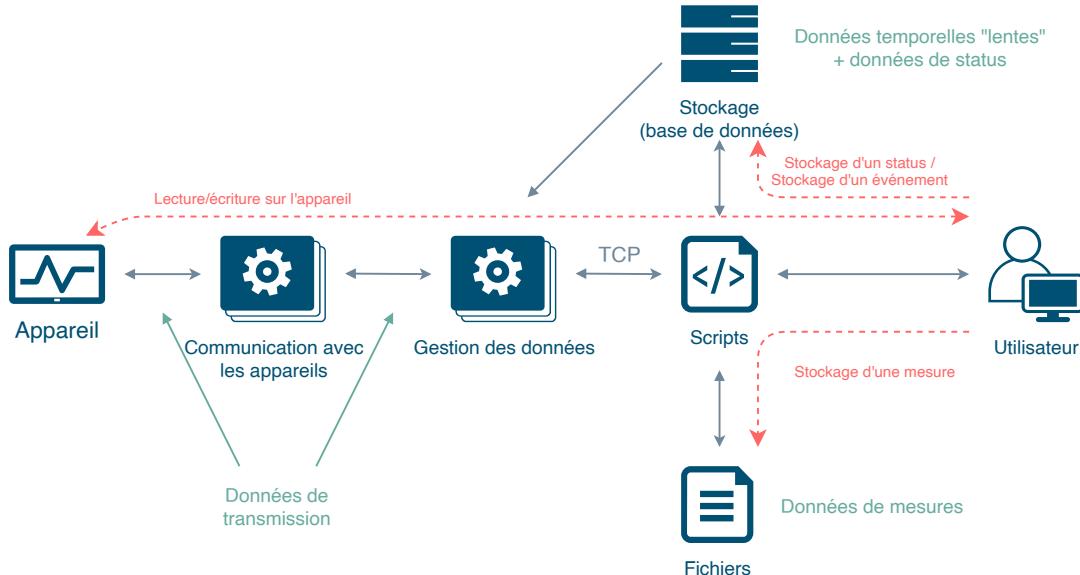


FIGURE 3.2 – Résumé des flux de données

Cette architecture répond aux contraintes suivantes :

- Communication avec les appareils gérée par le C++
- Possibilité de stocker et partager tous types de données simplement : les librairies Python permettent de stocker les mesures ou images sous formes de fichiers standards, qui peuvent être partagés facilement
- Prévision pour le datalogging : le stockage avec *timestamp* de l'objectif secondaire permet une gestion du temps transparente pour l'utilisateur.

3.2 Interaction avec l'utilisateur

L'objectif est de simplifier au maximum l'interaction avec l'utilisateur et d'atteindre un niveau d'abstraction le plus élevé possible lors de l'écriture des scripts. L'interface doit accompagner l'utilisateur pour la configuration des appareils (recherche des appareils disponibles, test de connexions etc...).

Les scripts doivent être réalisés le plus simplement possible sans nécessiter une connaissance de toute la chaîne de communication entre l'ordinateur et l'appareil.

Il est primordial que l'utilisateur ait une liberté totale lors de la création d'un banc de tests. Le projet doit être développé, afin que l'utilisateur puisse tirer parti de fonctions tierces, sans nécessiter de modifier le projet. Cette approche justifie l'utilisation de scripts, car ils suppriment les limitations d'une interface graphique (si une fonctionnalité n'est pas codée, elle n'est pas utilisable).

Une interface graphique reste toutefois nécessaire pour afficher l'état du système, la configuration des appareils ainsi que les paramètres de fonctionnement.

3.3 Programme C++

Le programme C++ est réalisé avec Qt ce qui permet d'utiliser :

- Le système de Signals/Slots[31] donne la possibilité de traiter facilement des événements asynchrones et des communication entre threads¹
- Les librairies Qt, qui apportent beaucoup de fonctionnalités (gestion des ports série, communication Ethernet, etc...)

Les points principaux du programme sont les suivants :

- Gestion des appareils : `devicesCollection` (voir 3.4)
- Interaction avec les scripts (canal TCP) : `scriptServer` (voir 3.5)
- Gestion des requêtes : `requestsManager` (voir 3.6)
- Lancement des scripts : `scriptsManager` (voir 3.7)
- Affichage des données : `storageManager` (voir 3.8)

Chacun de ces points sera représenté par une "super-classe" dans une classe maître "Core". La figure 3.3 montre un exemple de requête de lecture depuis un script python

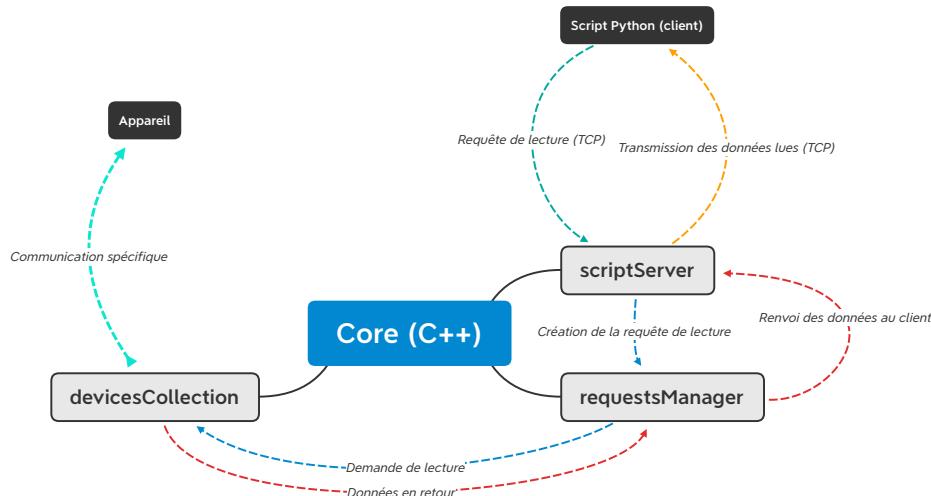


FIGURE 3.3 – Fonctionnement simplifié du programme avec exemple de lecture sur un appareil

Les traitillés représentent des couples signal/slot permettant de transmettre des événements et des données entre les différentes classes qui fonctionnent de manière asynchrone.

Cette organisation permet de séparer efficacement les fonctions principales du programme.

1. Plusieurs processus qui fonctionnent en parallèle dans un même programme

3.4 Gestion des appareils

Les rôles de cette partie sont :

- Enregistrement des appareils
- Création / suppression d'appareils
- Gestion des communications

Chaque appareil est désigné par une classe "device". Cette classe sert de base à une classe spécifique pour chaque type d'appareil (Ethernet, USB, GPIB ou série). La communication spécifique est traitée dans la classe device (et sa hiérarchie descendante).

Un appareil est référencé par :

- Son nom (Par exemple : "Oscilloscope Keysight")
- Son alias (Par exemple "osc0")

L'alias est utilisé pour référencer l'appareil dans le script.

3.4.1 Communication avec les appareils

La communication est traitées sur 2 couches :

- Type d'appareil
- Lien avec l'appareil : *link*

Le type d'appareil décrit la façon dont l'appareil est connecté à l'ordinateur. Le *link* implémente les fonctions pour communiquer avec l'appareil et, si nécessaire, les librairies spécifiques. Cette façon de séparer se retrouve dans l'organisation des classes du programme C++.

Chaque appareil peut posséder plusieurs *links* en fonction du protocole choisi par l'utilisateur. La figure 3.4 illustre les combinaisons de types et de leurs protocoles.

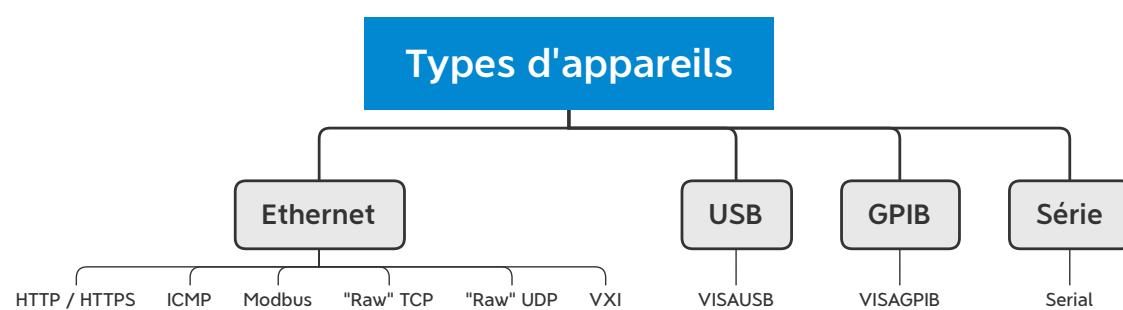


FIGURE 3.4 – Liste des protocoles pour chaque type

3.4.2 Ethernet

Les links suivants ont été réalisé pour le projet :

- HTTP / HTTPS : Possibilité d'envoyer et de recevoir des requêtes HTTP avec un serveur. Dans la pratique, cette fonctionnalité est utilisée principalement pour détecter si un serveur web est présent²
- ICMP : Utilisé uniquement pour effectuer une requête ping sur l'appareil (déttection de présence sur un réseau)
- Modbus : Protocole de communication très répandu[28]
- "Raw" TCP et "Raw" UDP : Transfert de données via les protocoles TCP ou UDP directement. Les données sont envoyées telles-quelles (comme sur un port série)
- VXI / LXI : Utilisation du protocole VXI sur Ethernet pour encapsuler les données des appareils de laboratoire

Tous les appareils connectés à Ethernet et testés lors du travail peuvent être contrôlés grâce à l'un de ces *links*.

Les librairies QTcpSocket[6] et QUdpSocket[6] sont utilisées pour assurer la connexion via TCP et UDP respectivement.

3.4.3 USB

Seuls les appareils USBTMC (appareils de laboratoire) sont configurés pour le projet. A cause de la difficulté du protocole USB, la librairie VISA[11] sera utilisé pour communiquer avec les appareils.

La librairie VISA permet de détecter automatiquement les appareils connectés par USB à l'ordinateur et prend en charge la communication.

3.4.4 GPIB

Comme l'interface GPIB est utilisée uniquement pour les appareils de laboratoire, il n'est pas nécessaire de gérer une communication universelle.

Comme pour la communication USB, la librairie VISA sera utilisée pour détecter et communiquer avec les appareils. La gestion de l'interface USB-GPIB se fait automatiquement dans VISA à condition que les drivers et/ou programmes spécifiques à l'interface soient installés.

Dans le cas de l'interface Agilent 82357B, il est nécessaire d'installer le programme *Keysight IO Librairies Suite*[1]

3.4.5 Série

Comme le protocole série ne définit pas la structure des données, il n'y a pas besoin de créer de links différents. Seul le link "série" sera fait (envoi de données brutes dans le port série).

La librairie QSerialPort[7, 8] permet d'effectuer la connexion, ainsi que la transmission de données.

2. Le protocole LXI oblige les appareils à créer un serveur web

3.5 Interaction avec les scripts

La communication entre les scripts et le programme C++ utilise un canal TCP. Cette option est la plus simple tout en étant indépendante du système d'exploitation.

L'adresse de connexion locale³ est `localhost` et le port est 8445⁴.

Les commandes commencent par un identifiant sur 1 byte puis :

- Des valeurs de taille fixe à la suite
- Des valeurs de taille variable précédées par leur taille sur un nombre de bytes fixe

Type	Données de taille fixe (8 bytes)								Données de taille variable (N bytes)								...
	A_0	A_1	A_2	\dots	A_6	A_7		N	B_0	B_1	\dots	B_{N-2}	B_{N-1}				...

TABLE 3.1 – Exemple de commande

Les commandes principales et leurs arguments sont :

1. Requête de lecture sur un appareil
 - Alias de l'appareil
2. Requête d'écriture sur un appareil
 - Alias de l'appareil
 - Données
3. Demande de driver pour un appareil
 - Alias de l'appareil
4. Demande de chemin de la base de donnée
 - aucun argument

3.6 Gestion des requêtes

Toutes les actions effectuées par l'utilisateur qui nécessitent une communication avec un appareil sont traitées par le gestionnaire de requêtes.

Ce système est mis en place afin de laisser le temps à chaque appareil d'effectuer la mesure / acquisition avant de retourner les valeurs. Le gestionnaire de requêtes crée une liste d'attente des opérations (lecture ou écriture) à effectuer sur chaque appareil.

Cette liste est visible dans l'onglet "Diagnostic" de l'interface graphique sous forme d'un tableau, avec autant de lignes que d'appareils.

3.7 Lancement des scripts

Les scripts peuvent être exécutés depuis l'interface (mais pas exclusivement). Comme la communication entre le script et le programme C++ est faite par TCP, il suffit que le script se trouve sur le même ordinateur pour que la communication fonctionne.

3. Sur le même ordinateur
4. Choisi arbitrairement

3.8 Données

Comment mentionné dans la partie 2.5, les données sont stockées à des endroits différents en fonction de leur type. La super-classe `storageManager` se charge des chemins de stockage (notamment l'emplacement de la base de données). Le script Python effectue une requête pour l'emplacement de la base de données au démarrage du script, puis ouvre la base de donnée en écriture. La base de donnée est ouverte en lecture par le programme C++, uniquement lorsque le script envoie une commande de mise à jour.

3.8.1 Blocs

Un "bloc" est une unité de stockage créé pour rendre transparente la création, la modification et la suppression des tables dans une base de donnée (voir 2.5.4). L'utilisateur peut créer, modifier ou supprimer les blocs à partir du script Python.

Les blocs sont conçus pour fonctionner avec une colonne de temps (*timestamp*), puis des colonnes de données (A, B, ..., AA, AB, etc...). Lorsqu'un enregistrement est ajouté et si le temps n'est pas spécifié, c'est l'heure actuelle qui sera stockée dans la colonne timestamp. La table 3.2 montre un exemple de bloc.

Timestamp	A	B	C
1625316368	0	20.5	5
1625316369	8	20.4	5
1625316370	21	20.5	5
1625316371	37	20.6	6
:	:	:	:
1625316526	100	20.3	6

TABLE 3.2 – Exemple de bloc

Les données temporelles "lentes"⁵, les événements et les statuts sont stockés ensemble dans les blocs.

Le timestamp[9] représente une date en nombre de secondes depuis le 1er janvier 1970. Il est facile de convertir ce nombre en heure sous la forme "05.07.2021 18h30"

Les colonnes A, B et C contiennent des exemples de données qui varient en fonction du temps

- A : Progression de 0% à 100%
- B : Température
- C : Nombre de circuits électroniques testés dans un robot

Il est également possible de créer un bloc pour chacune de ces valeurs. Elles sont regroupées pour l'exemple

5. Avec une période inférieure ou égale à 1 seconde

3.8.2 Affichage des données

La table 3.3 désigne où sont stockées les données et comment elles sont présentées à l'utilisateur

Type de données	Stockage	Affichage	Remarque
Console système ⁶	Pas de stockage	Onglet "Diagnostique"	
Console client (bloc "console")	Base de données	Onglet "Scripts"	
Blocs	Base de données	Onglet "Données"	
Images ⁷	Système de fichier (.png, .bmp)	Pas d'affichage prévu	Responsabilité de l'utilisateur
Oscillogrammes	Système de fichier (.csv, .dat)	Pas d'affichage prévu	

TABLE 3.3 – Emplacement et affichages des types de données

Grafana

L'objectif secondaire du stockage dans la base de données et l'utilisation de Grafana est réalisé. L'utilisation du logiciel est décrit dans le manuel d'utilisation (voir annexe B). Des exemples de commandes SQL pour chaque bloc sont disponibles avec un double-clic sur le bloc en question (dans l'interface graphique).

L'utilisateur reste libre de modifier les requêtes SQL pour obtenir un fonctionnement spécifique.

3.9 Projets

Un gestionnaire de projet permet d'organiser les scripts ainsi que le stockage pour chaque utilité. La figure 3.5 illustre comment s'organisent les projets sur l'ordinateur de contrôle.

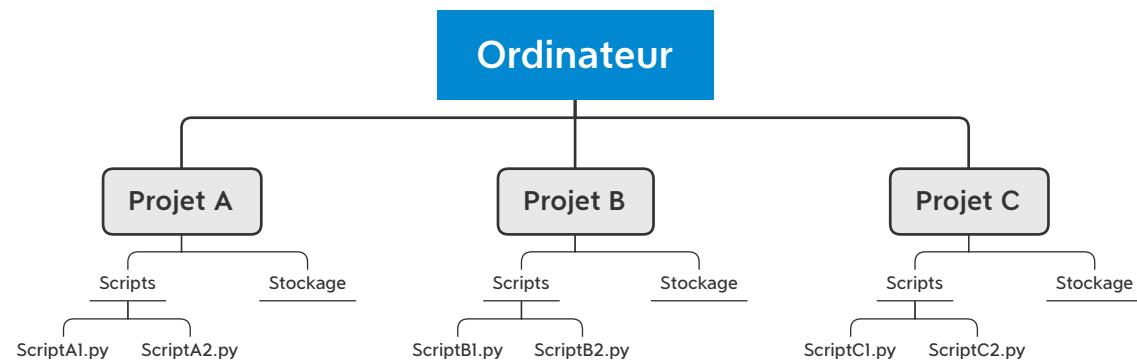


FIGURE 3.5 – Organisation des projets

6. Console du programme C++

7. Par exemple, des captures d'écran d'appareils

3.10 Interface graphique

L'interface graphique est réalisée avec le langage QML en exploitant un système modulaire : la fenêtre de base `mainwindow.qml` instancie des blocs comme la liste des appareils (`Devices.qml`) ou l'affichage des données (`Data.qml`). Cette hiérarchie continue à chaque fois qu'il est nécessaire de séparer des fonctionnalités, d'ordonner le code ou de réutiliser des blocs.

3.10.1 Organisation

Lorsque le programme est démarré, une fenêtre de dialogue propose à l'utilisateur de choisir un projet existant ou d'en créer un nouveau

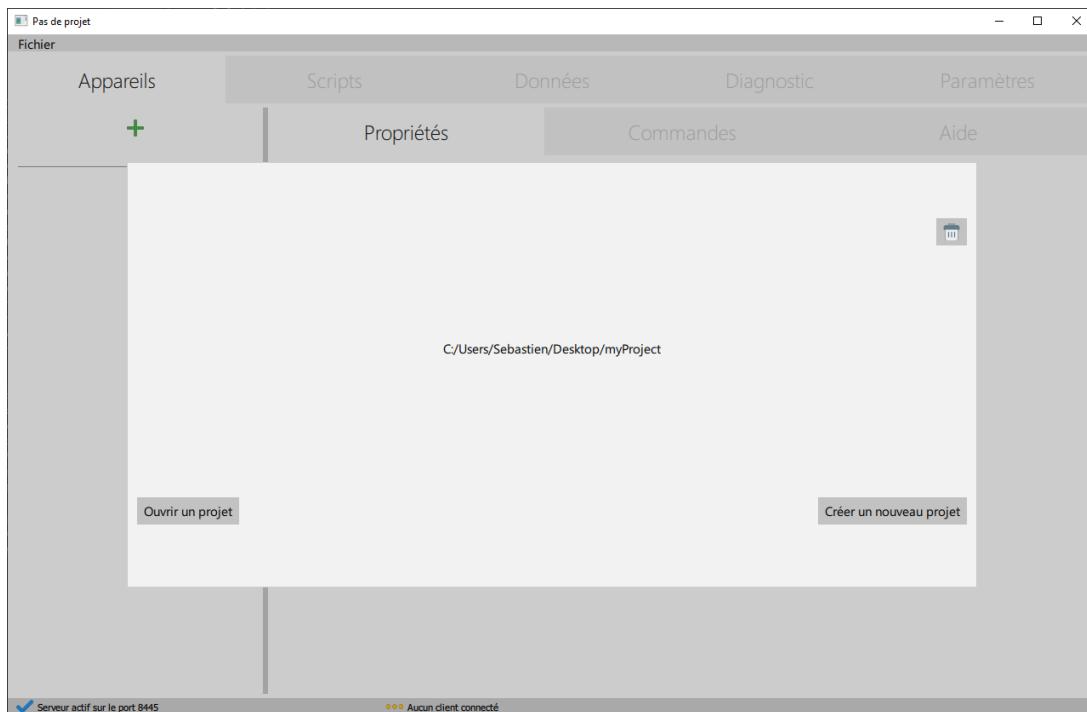


FIGURE 3.6 – Écran de démarrage

Le projet fermé précédemment s'ouvrira lors du prochain démarrage.

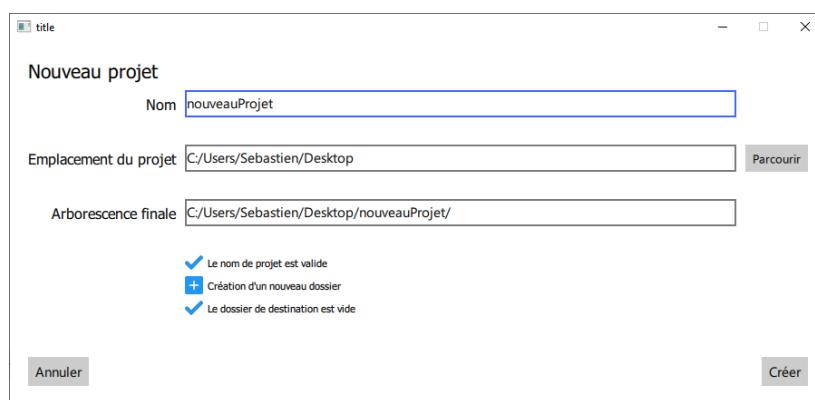


FIGURE 3.7 – Nouveau projet

le programme est divisé en 5 onglets :

1. Appareils
 - Liste des appareils
 - Propriétés
 - Commandes disponibles pour chaque appareil
 - Aide (affichage du fichier d'aide)
2. Scripts
 - Liste des scripts
 - Lancement d'un script
 - Console d'exécution
 - Console client (bloc "console")
3. Données
 - Liste des blocs
4. Diagnostic
 - Affichage des requêtes en cours
 - Console système
5. Paramètres
 - Dossiers de recherche des drivers
 - Liste des drivers
 - Vérification des installations (Python et librairie)

Appareils

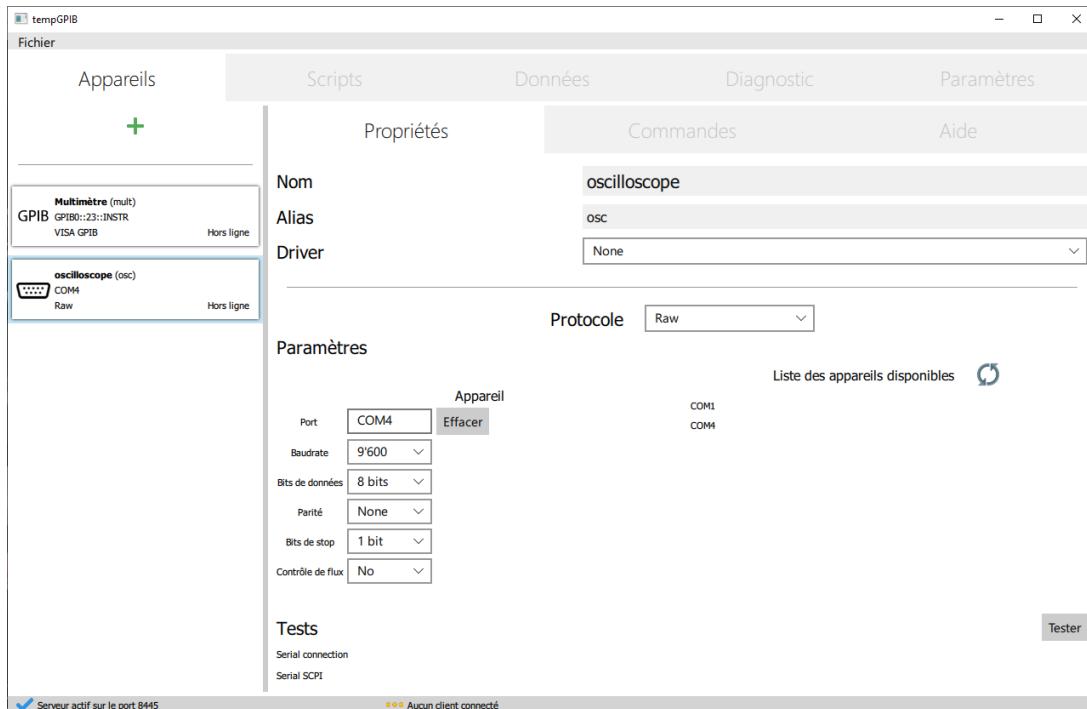


FIGURE 3.8 – Onglet "Appareils"

Lorsqu'un appareil est sélectionné, l'onglet des propriétés s'active et il est possible de configurer l'appareil. Il est également possible d'accéder aux commandes de l'appareil pour autant qu'un driver ait été configuré pour cet appareil. L'onglet "Aide" permet d'afficher un document écrit en Markdown[21] pour chaque driver. Le document doit porter le même nom que le driver pour qu'il soit détecté par le programme (à l'exception de l'extension .py qui devient .md)

Scripts

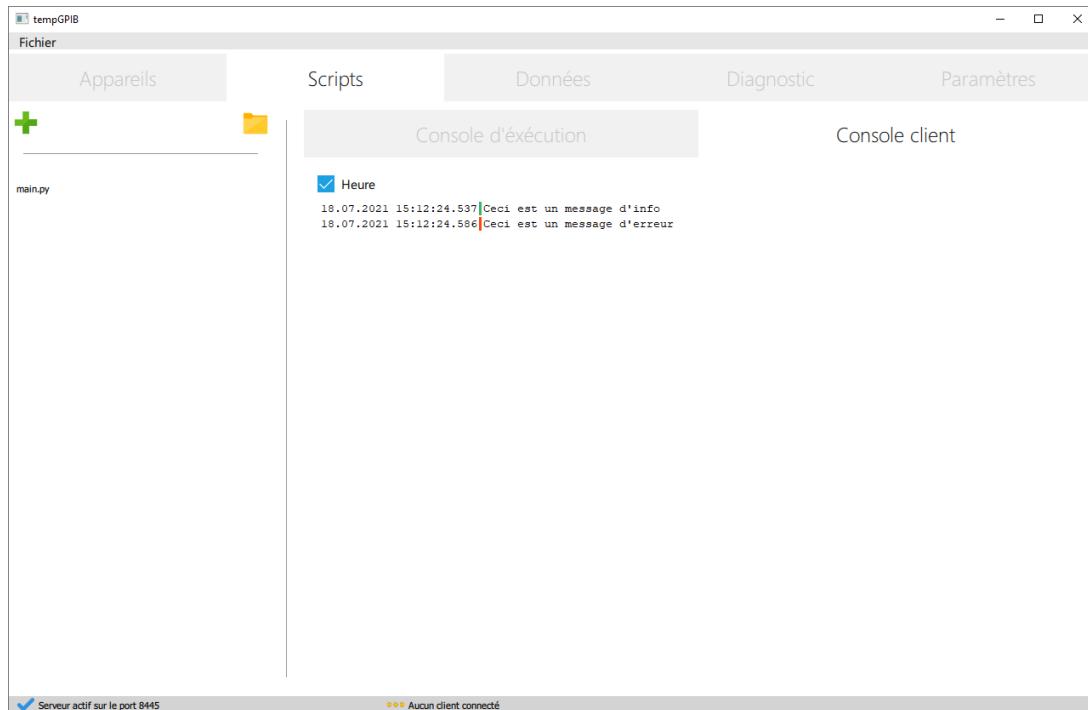


FIGURE 3.9 – Onglet ”Scripts”

La liste des scripts sur la gauche est mise à jour automatiquement. Un bouton ”Lancer le script” permet de l’exécuter. La console d’exécution affiche les données de la sortie standard. La console client affiche le contenu du bloc ”console” qui est mis à jour dans le script (par les commandes `console.logX`).

L’exemple de la figure 3.9 illustre une écriture dans le bloc `console` par le script. Le script est le suivant :

```

1 | from atmp import console
2 |
3 | console.logInfo('Ceci est un message de type info')
4 | console.logError('Ceci est un message de type erreur')
```

Données

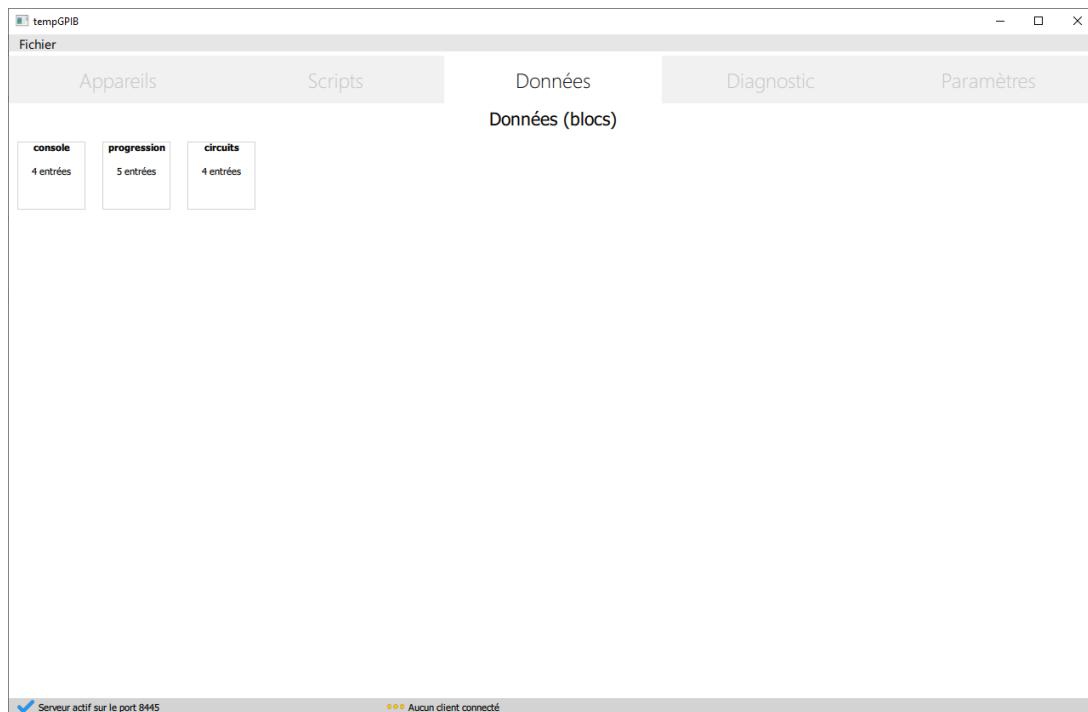


FIGURE 3.10 – Onglet ”Données”

L’onglet de données affiche une liste des différents blocs créées. La figure 3.10 montre trois exemples de blocs :

- *console* : Liste des événements enregistrés
- *progression* : Progression du script (en pourcentage)
- *circuits* : Nombre de circuits testés

Diagnostic

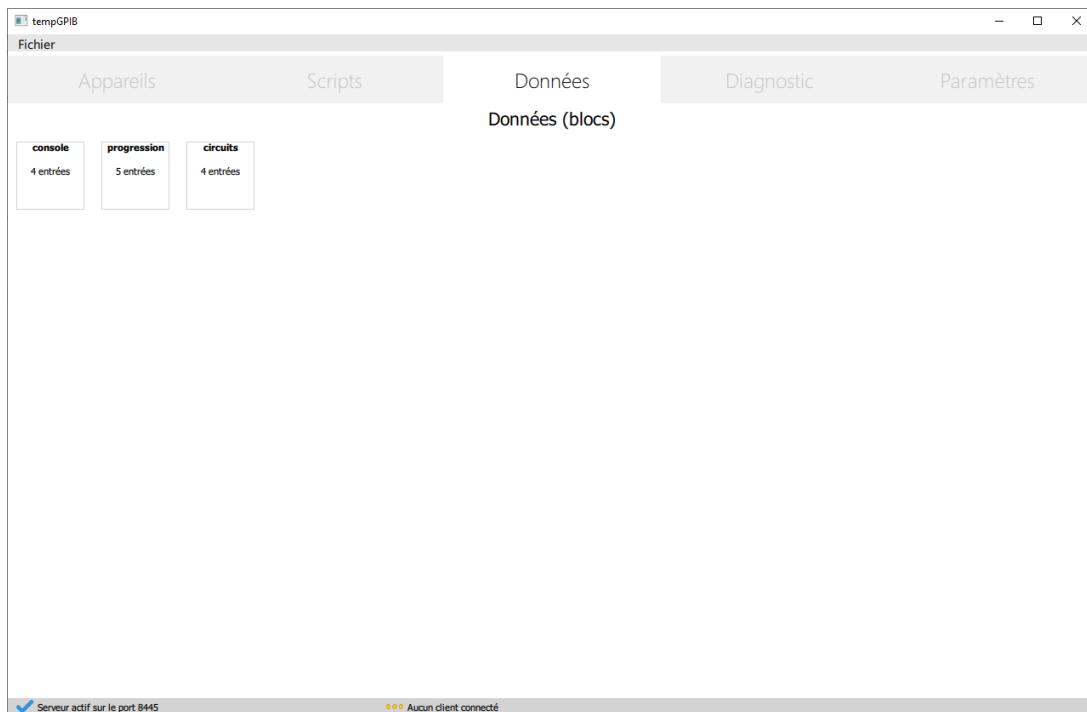


FIGURE 3.11 – Onglet ”Diagnostic”

Une liste des appareils (avec leurs alias) est visible sur la gauche de la figure 3.11. Les requêtes de lecture / écriture s'affichent à droite des appareils en formant une liste (en fonction de leur heure d'émission).

La console de droite indique toutes les opérations effectuées par le programme, avec une coloration en fonction de la gravité de l'événement

Paramètres

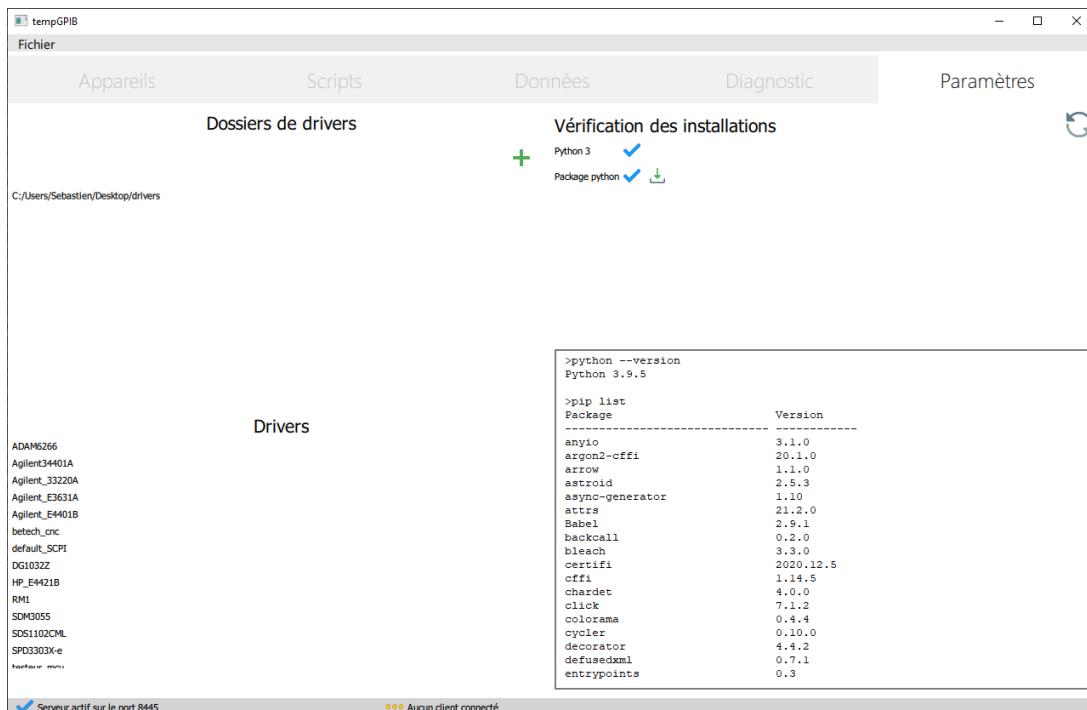


FIGURE 3.12 – Onglet ”Paramètres”

Les dossiers de recherche des drivers sont listés sur la gauche de la figure 3.12. Tous les drivers détectés sont affichés en bas dans la catégorie ”Drivers”.

La partie de droite résume l’état de l’installation de Python et de la librairie spécifique (voir 3.11).

3.11 Package Python

Le package Python pyatmp (Python Automated Test & Measurement Platform) permet d'installer les fichiers nécessaires à partir de PyPi[29] avec une seule commande pip⁸. La commande d'installation du package est la suivante

```
pip install pyatmp
```

Le package contient un module "atmp", qui lui-même contient les fichiers pour la communication avec le programme C++.

3.11.1 Drivers

Un driver est une classe qui est instanciée à chaque fois qu'un appareil est créé.
Chaque driver doit être de la forme

```
1  """
2  Commentaire (facultatif)
3  """
4
5  class new():
6      def __init__(self, device):
7          self.device = device
8
9      def writeRaw(self, data : bytearray):
10         self.device.write(data)
11
12     def readRaw(self):
13         return self.device.read()
```

Listing 3.1 – Exemple de driver

8. Pip[26] permet d'installer des packages Python automatiquement

3.12 Licences

Plusieurs librairies ou travaux de tiers sont utilisés dans ce projet (voir table 3.4). Sauf exception, les classes Qt seront regroupées sous le nom "classes Qt".

Nom	Auteur	Licence
Classes Qt	Qt	LGPL[18]
VISA	IVI Foundation &	Pas de licence nécessaire

TABLE 3.4 – Liste des librairies utilisées

La licence LGPL[18] permet la modification, la distribution, l'utilisation privée et commerciale d'un travail. Cette licence est une version moins restrictive de la licence GPL[17].

La librairie VISA est gérée par la Fondation IVI[10] mais le téléchargement est fait via des distributeurs. Ces derniers peuvent ajouter une surcouche sur la librairie VISA (comme National Instruments[22]), ce qui peut modifier la licence. La librairie VISA de National Instruments a été utilisé au début du projet, avant d'être abandonnée, car son utilisation ne respectait pas les conditions de licence.

Le site de Keysight[11] mentionne "Ne nécessite pas de licence" (licence payante). Le fichier de licence ne mentionne aucune restriction, elle a donc été utilisée.

La licence open-source de Qt (LGPL) permet une utilisation libre de toutes les librairies de l'environnement.

3.12.1 Choix de licence pour le travail

Au moment de l'écriture de ce document, le travail est sous licence GPL[17], ce qui le rend open-source avec une obligation de documentation des modifications.

Ce choix permet de promouvoir l'amélioration du travail dans le futur.

Chapitre 4

Application

L'objectif est de reprendre le système de banc de test réalisé précédemment et de l'adapter avec la nouvelle méthode. Un projet et des drivers spécifiques seront réalisés pour le banc de test.

4.1 Module RM1S2

Le module à tester est le RM1S2. C'est un module de communication radio capable de fonctionner selon plusieurs modulations différentes (voir table 4.1).

value	bitrate [bps]	fréquence du carré ¹	type	déviation
0	2400	1200	2GFSK	1250
1	4800	2400	2GFSK	2500
2	9600	4800	2GFSK	5000
3	19200	9600	2GFSK	10000
4	38400	19200	2GFSK	20000
5	60000	30000	2GFSK	30000
6	100000	50000	2GFSK	50000
7	50000	-	4GFSK	21000
8	100000	-	4GFSK	42000
9	200000	-	4GFSK	82700
10	120000	-	MSK	60000

TABLE 4.1 – Liste des modulations disponibles

La fréquence de la porteuse est de 869.525 MHz

La communication entre le module et l'hôte se fait à l'aide de commandes de la forme

N	Type	DataN-2	DataN-3	...	Data1	Data0
---	------	---------	---------	-----	-------	-------

Avec N le nombre de bits qui suivent.

1. Fréquence d'un signal carré permettant une modulation FSK sur la porteuse

4.2 Protocole

Le banc de test doit :

1. Initialiser les instruments et le robot
2. Placer la tête de mesure sur le premier circuit
3. Programmer le circuit (flash du firmware)
4. Ajuster la fréquence de la porteuse (facultatif, car les TCXU utilisés sont suffisamment précis)
5. Effectuer une mesure de la puissance d'émission
6. Effectuer une mesure du Bit Error Rate
7. Tester la communication entre deux modules

4.2.1 Initialisation

Tous les appareils doivent être allumés manuellement avant le démarrage du script.

Pour pouvoir utiliser le robot, il est nécessaire d'effectuer une séquence de "homing", c'est à dire un référencement de la position des axes. Pour lancer le homing, la commande \$H est utilisée (avec un caractère de terminaison \n). Une fois le homing terminé, le robot est placé dans le coin en haut à droite.

L'alimentation est reliée et configurée selon la figure 4.1.

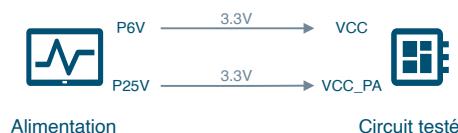


FIGURE 4.1 – Configuration de l'alimentation

4.2.2 Déplacement du robot

Il existe deux méthodes pour déplacer le robot :

1. Déplacement avec le référentiel machine (G53). Les coordonnées sont référencées sur le homing
2. Déplacement avec un référentiel de travail (G54). Le référentiel de travail permet de déplacer le robot par rapport à un point de départ (par exemple la position du premier circuit)

Les deux méthodes utilisent les mêmes unités de distance.

La première méthode sera utilisée par souci de simplicité. Le traitement des coordonnées peut facilement être traité dans le script Python.

Il a toutefois été noté que le positionnement de la tête sur les circuits est très difficile : les pogo-pins n'appuient pas correctement sur les pads et le contact se fait sur l'arrêté des perçages du circuit imprimé (voir figure 4.2)

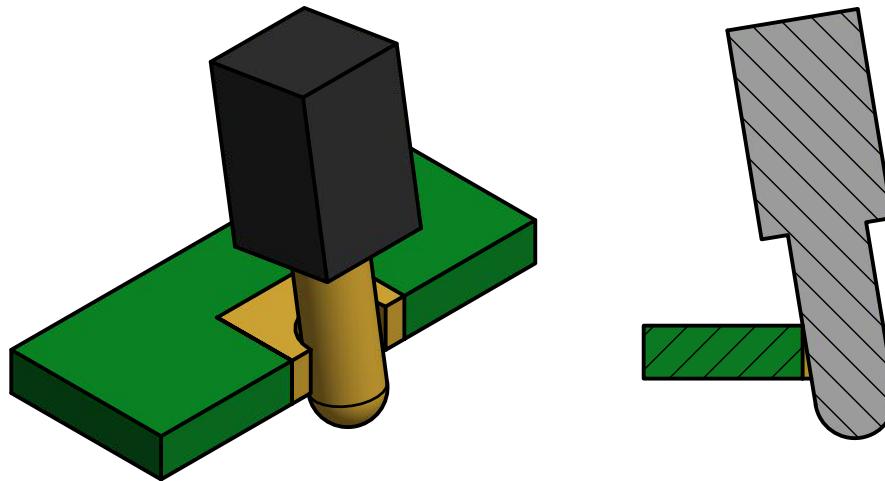


FIGURE 4.2 – Contact entre le circuit et les pogo-pins

Les pogo sont brasés avec un angle sur la tête de mesure et le contact se fait grâce à la souplesse de cette connexion. Afin de démontrer au mieux le fonctionnement du projet, un seul circuit sera testé et les mouvements du robot ne seront pas nécessairement relancés à chaque démarrage du script (robot déjà placé sur le circuit).

4.2.3 Programmation

Le circuit utilise un STM32. Pour y charger le firmware (fichier .bin), il faut utiliser l'outil STM32CubeProgrammer[27]. Afin d'automatiser le processus, il est possible d'utiliser l'outil en ligne de commande (**STM32_Programmer_CLI.exe**). L'outil doit être installé depuis le site du fabricant puis il est possible d'appeler la ligne de commande à partir de son chemin, par exemple :

`C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\STM32_Programmer_CLI.exe`
La commande pour flasher le firmware est la suivante (compatible pour Windows).

```
"C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer
\bin\STM32_Programmer_CLI.exe" -c port=COM9 -w "RM1S2_HW4.2D_FW1.1.9_STM32L443CC.bin" 0x08000000
```

- `C:\Program Files\...\STM32_Programmer_CLI.exe` : Chemin de l'exécutable
- `-c port=COM9` : Port série à utiliser pour la programmation (peut être changé)
- `-w "RM1S2_HW4.2D_FW1.1.9_STM32L443CC.bin"` : fichier de firmware
- `0x08000000` : Adresse mémoire de début (toujours la même)

4.2.4 Mesure de la puissance d'émission

La puissance d'émission est mesurée en fonction de la tension d'alimentation du PA. Pour cela, sa tension est ajustée entre un minimum et un maximum par pas configurables. Pour chaque pas, la puissance est mesurée à l'aide de l'analyseur de spectre.

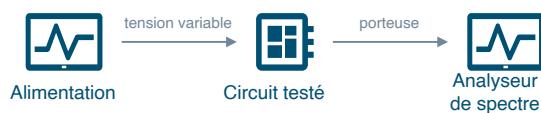


FIGURE 4.3 – Schéma de la mesure de la puissance d'émission

4.2.5 Mesure du Bit Error Rate

La mesure du Bit Error Rate se fait dans le circuit lui-même. Pour cela il faut :

1. Activer les commandes de test
2. Générer une porteuse modulée à la bonne fréquence (voir table 4.1)
3. Demander une mesure du BER² au module
4. Recommencer pour une autre fréquence de transfert ou une autre puissance d'émission



FIGURE 4.4 – Schéma de la mesure du BER

Il est possible de configurer le module afin d'accepter plusieurs modulations différentes. La table 4.1 liste les différentes modulations. La modulation est changée avec la commande

0x02 0x62 ...

Avec les “...” remplacés par le champ “value” (voir table 4.1). Seules les modulations 2GFSK seront testées.

4.2.6 Test de communication entre deux modules

Le test de communication n'a pas été réalisé pendant ce travail, car il n'existe pas de deuxième module prêt pour effectuer une communication avec le premier.

Le schéma de mesure serait le suivant

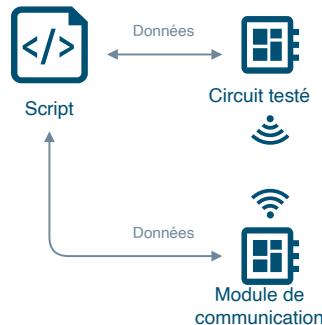


FIGURE 4.5 – Schéma de communication entre deux modules

2. Bit Error Rate

4.2.7 Script

Le script est séparé en modules, afin de le rendre plus lisible

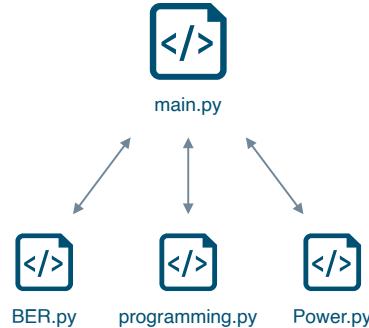


FIGURE 4.6 – Organisation du script de banc de test

```

1  # Protocole de banc de test pour RM1S2 RF4.2D
2  # 07.07.2021
3  # Sébastien Deriaz
4
5  # Librairies standards
6  import time
7  import easygui
8  # Librairie de contrôle spécifique
9  from TCPLib import devices, blocks, console
10 # Modules
11 import BER
12 import Power
13 import programming
14
15 # Effacement de la console de log
16 console.clear()
17 console.logInfo("Démarrage du testbench")
18
19 # Appareils
20 # Analyseur de spectre
21 sa = devices.device('E4401B')
22 # Générateur RF
23 genrf = devices.device('E4421B')
24 # Générateur de signal (modulation du signal RF)
25 gen = devices.device('33220A')
26 # Alimentation de laboratoire (alimentation du circuit)
27 ps = devices.device('E3631A')
28 # Robot (CNC)
29 cnc = devices.device('cnc')
30 # Module radio (RM1S2)
31 rm = devices.device('rm')
32 # Testeur (Carte blanche)
33 tester = devices.device('testeur')
34
35 ## Initialisation du robot et des instruments
36
37 # Le homing de la machine n'est pas effectué car il n'est pas possible, pour
38 # le moment, de se placer de manière fiable et répétable sur un circuit
39 # Le robot doit être placé manuellement sur le circuit
40 #
41 # if(easygui.ynbox("Effectuer un homing de la machine ? ")):
42 #     pass
43 # Initialisation
44 # Homing du robot
45 # cnc.init()
46 # Position pour atteindre le circuit X2 Y0
47 #cnc.moveAbsoluteXY(-128.8, -253.4)
48 # cnc.moveZ(-2) # Contact avec le circuit
49
  
```

```
50 | ps.setChannelLimits('P6V', voltage=3.3, current=0.1)
51 | ps.setChannelLimits('P25V', voltage=3.3, current=0.1)
52 | ps.setOutputsState(True)
53 | time.sleep(1)
54 | # Mesure du courant dans le circuit
55 | console.logInfo("Contact avec le circuit, Courant dans P6V : %.2fmA, Courant dans P25V : %.2
56 |   fmA" % (ps.getCurrentMeasurement(
57 |     'P6V')*1000, ps.getCurrentMeasurement('P25V')*1000))
58 |
59 | ## Programmation et récupération de l'identifiant
60 | console.logInfo('Programmation du circuit...')
61 | id = programming.program(rm, tester, ps)
62 | console.logInfo("Identifiant : %s" % id)
63 |
64 | ## Mesure de la puissance d'émission
65 | console.logInfo('Mesure de la puissance')
66 | Power.powerMeasurement(genrf, sa, ps, rm, 'Power.png')
67 |
68 | #210711 - Inversion BER <-> Power, Vérifier le bon fonctionnement du testbench
69 | ## Mesure du Bit Error Rate
70 | console.logInfo('Mesure du Bit Error Rate')
71 | BER.BERMeasurement(rm, gen, genrf, sa, ps, 'BER.png')
72 |
73 | ## Arrêt du testbench
74 | ps.setOutputsState(False)
75 | console.logInfo("Fin du testbench")
```

Listing 4.1 – Script principal (main.py)

```
1 import os
2 import time
3
4
5 def program(rm, tester, ps):
6     # Fermeture du port série (pour laisser le programmeur prendre la main)
7     rm.close()
8
9     # Activation du mode programmation
10    tester.setProgrammingMode()
11
12    # Activation de l'alimentation
13    ps.setOutputsState(True)
14
15    # Programmation du module
16    exePath = r"C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\
17    STM32_Programmer_CLI.exe"
18    binPath = r"RM1S2_HW4.2D_FW1.1.9_STM32L443CC.bin"
19    command = "\"%s\" -c port=COM9 -w \"%s\" 0x08000000" % (exePath, binPath)
20    stream = os.popen(command)
21    output = stream.read()
22
23    # Attendre 2 secondes puis reset le module
24    time.sleep(2)
25    ps.setOutputsState(False)
26
27    # Mettre le système en mode standard puis démarrer le module
28    tester.setStandardMode()
29    ps.setOutputsState(True)
30
31    # Laisser le module démarrer puis demander son identifiant
32    time.sleep(2)
33    identifier = rm.getIdentifier()
34    return identifier
```

Listing 4.2 – Script de programmation (programming.py)

```

1 import numpy as np
2 import time
3 import matplotlib.pyplot as plt
4
5 def BERMeasurement(rm, gen, genrf, sa, ps, image=None):
6     """
7         rm : Remote module
8         gen : low frequency generator
9         genrf : RF generator
10        sa : Spectrum analyzer
11        ps : power supply
12    """
13
14    # Configuration de l'alimentation (3.3V sur chaque canal)
15    ps.setChannelLimits('P6V', voltage=3.3, current=0.05)
16    ps.setChannelLimits('P25V', voltage=3.3, current=0.05)
17    ps.setOutputsState(True)
18    time.sleep(1)
19
20    # Connexion au module
21    id = rm.getIdentifier()
22    if(id is None):
23        raise Exception("Could not connect with radio module")
24
25    rm.enableTestCommands()
26
27    # Configuration des générateurs (porteuse modulée FSK)
28    gen.selectFunction('SQU')
29    gen.setAmplitude(2)
30    gen.setOffset(0)
31    gen.setOutputState(True)
32    genrf.setCarrierFrequency(869.525e6)
33    genrf.setFMModulationSource(1, 'EXT1')
34    genrf.setFMDeviation(modulator=1, deviation=1250)
35    genrf.setFMModulationState(1, True)
36    # Activation de la sortie RF
37    genrf.setModulationState(True)
38    genrf.setRFOutputState(True)
39
40    power = np.linspace(-110, -40, 10)
41    bitrates = np.array([2400, 4800, 9600, 19200, 38400, 60000, 100000, ])
42
43    M = np.zeros((bitrates.size, power.size, 3))
44    N = 1000
45    # Mesure et stockage des valeurs dans une matrice M
46    for i, br in enumerate(bitrates):
47        rm.setBitrate(i)
48        gen.setFrequency(br / 2)
49        for j, p in enumerate(power):
50            genrf.setCarrierPower(p)
51            time.sleep(0.5)
52            rawBER = rm.getBER(N)
53            ber = rawBER / N
54            M[i, j, :] = [br, p, ber if ber <= 0.5 else 1-ber]
55
56    # Si un chemin d'image est spécifié, l'image est enregistrée
57    if(not image is None):
58        fig, ax = plt.subplots(figsize=(15, 10))
59        for i in np.arange(M.shape[0]):
60            ax.plot(M[i, :, 1], M[i, :, 2], label='%d bit/s' % M[i, 0, 0])
61        ax.legend()
62        ax.set_xlabel('Puissance [dbm]')
63        ax.set_ylabel('BER [%]')
64        ax.grid()
65        fig.savefig(image)

```

Listing 4.3 – Script de mesure du BER (BER.py)

```

1 import numpy as np
2 import time
3 import matplotlib.pyplot as plt
4
5
6 def powerMeasurement(genrf, sa, ps, rm, image=None):
7     """
8         genrf : RF generator
9         sa : Spectrum analyzer
10        ps : Power supply
11        rm : Radio module
12    """
13    genrf.setRFOutputState(False)
14
15    ps.setChannelLimits('P6V', voltage=3.3, current=0.1)
16    ps.setChannelLimits('P25V', voltage=3.3, current=0.1)
17    ps.setOutputsState(True)
18    time.sleep(1)
19
20    # Configuration de l'analyseur de spectre
21    sa.setCenterFrequency(869.525e6)
22    sa.setAutoFrequencyStepSize(True)
23    sa.setFrequencySpan(10e3)
24
25    # Configuration du module
26    rm.enableTestCommands()
27    rm.setRFTestMode(1) # émission de la porteuse
28
29    voltage = np.linspace(2.5, 3.3, 20)
30    voltageMCU = voltage
31    amplitudes = np.zeros_like(voltage)
32
33    # Lancement de la mesure et stockage dans une matrice M
34    for i, v in enumerate(voltage):
35        ps.setChannelLimits('P6V', voltage=voltageMCU[i], current=0.1)
36        ps.setChannelLimits('P25V', voltage=v, current=0.1)
37        time.sleep(1)
38        sa.placeMarkerOnPeak(marker=1)
39        amplitudes[i] = sa.getMarkerAmplitudeValue(marker=1)
40
41    M = np.stack((voltage, amplitudes), axis=1)
42    # Si un chemin est spécifié, l'image est enregistrée
43    if(not image is None):
44        fig, ax = plt.subplots(figsize=(10, 10))
45        ax.plot(M[:, 0], M[:, 1])
46        ax.grid()
47        ax.set_xlabel('Tension (PA)')
48        ax.set_ylabel('Puissance [dbm]')
49        fig.savefig(image)

```

Listing 4.4 – Script de mesure de la puissance (Power.py)

4.3 Mesures

Les mesures effectuées sur le circuit sont représentées sous forme de graph et sont enregistrées au format .png par le script (à l'aide de la librairie matplotlib). Ces mesures servent principalement à démontrer le bon fonctionnement du script (récupération, traitement et affichage des données).

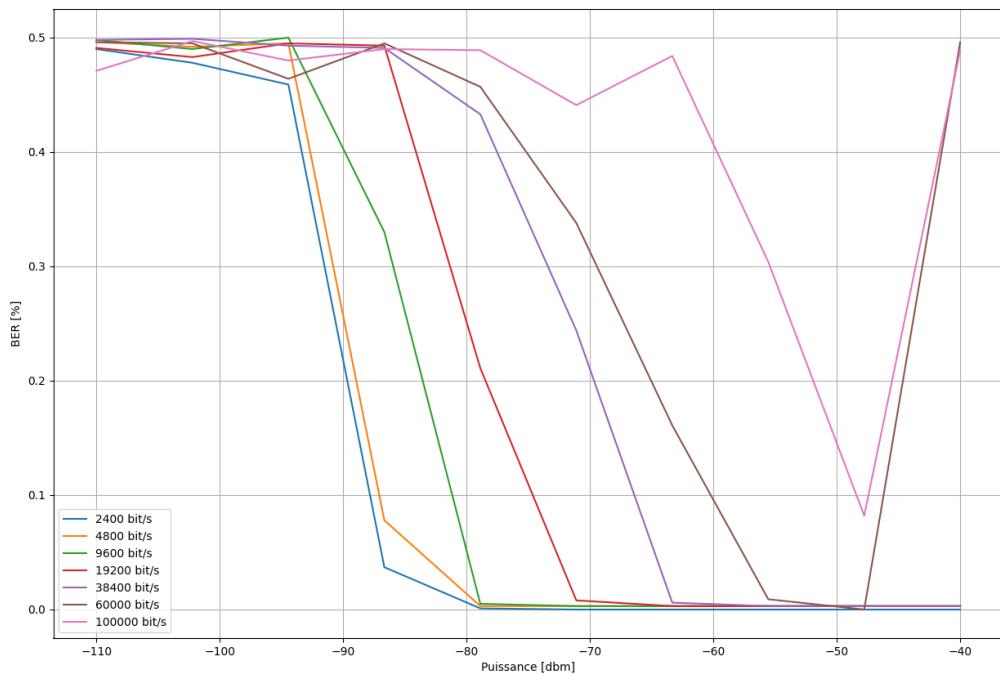


FIGURE 4.7 – Mesure du BER

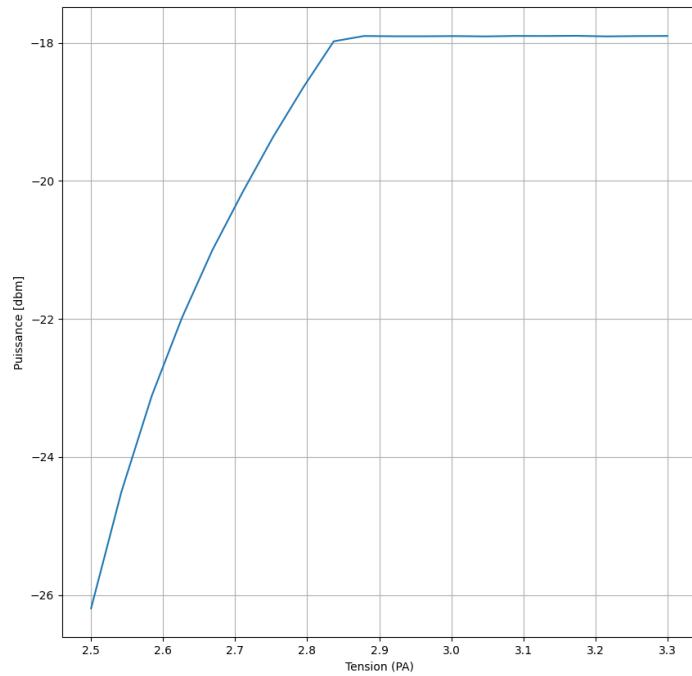


FIGURE 4.8 – Mesure de la puissance

Les courbes du Bit Error Rate correspondent aux prévisions, soit une augmentation de l'erreur lorsque la puissance et/ou le bitrate augmentent. Il semble toutefois qu'une augmentation de l'erreur apparaisse avec un bitrate et une puissance élevés. La mesure a été répétée plusieurs fois et le même phénomène s'est reproduit.

La mesure de la puissance montre une chute linéaire à partir d'une tension d'alimentation de 2.85 V, ce qui est attendu et correspond avec les valeurs données dans la datasheet du PA.

Chapitre 5

Conclusion

5.1 Cahier des charges

5.1.1 Objectifs primaires

L'objectif initial (mise en œuvre du robot de test de circuits électroniques) a été atteint. Toutes les fonctionnalités peuvent être réalisées grâce à la plateforme et de nouvelles pourraient facilement être ajoutées par la suite. Des tests ont pu être réalisés sur l'un des circuits et démontrent le succès de la communication avec les appareils et le circuit.

L'approche choisie a permis d'améliorer la polyvalence du système et d'étendre son utilisation au delà de l'objectif initial. Un grand nombre d'appareils ont été testés au cours du développement, sans déceler de problèmes.

Les objectifs secondaires ayant été réalisés, la plateforme est capable d'effectuer des tâches de monitoring sur de longues périodes. La fiabilité du système pour un fonctionnement longue durée n'a pas été testée.

5.1.2 Objectifs secondaires

L'outil Grafana s'est montré particulièrement efficace pour :

1. Afficher des logs (liste des événements)
2. Indiquer une progression (gauge)
3. Tracer l'évolution d'une variable en fonction du temps (graph)

Comme Grafana fonctionne sur un serveur web, il est particulièrement utile d'y accéder depuis un autre ordinateur pour observer le fonctionnement du système à distance.

5.2 Plateforme

5.2.1 Appareils

La communication avec les appareils Ethernet et série fonctionne parfaitement grâce aux librairies Qt.

La communication avec les appareils GPIB et USB (USBTMC) nécessite la librairie VISA[11] (utilisation de fichiers .h, .lib et .dll). VISA est disponible sur le site de plusieurs distributeurs (National Instruments, Keysight, Rhode & Schwarz), mais les conditions et le contenu varient. La version de Keysight a été utilisée, mais une analyse plus approfondie des termes de la licence pourrait être faite pour en valider l'utilisation (voir annexe D).

Tous les tests effectués au cours du travail ont montré une parfaite communication entre l'ordinateur et les appareils (aucunes données perdues, pas d'erreurs de transmission, etc...).

5.2.2 Drivers

Des drivers pour de multiples instruments ont été réalisés. La création s'est montrée simple, mais chronophage, car il existe un grand nombre de commandes pour chaque appareil. Il est toutefois possible de se concentrer sur les plus utilisés.

5.2.3 Scripts

Python est extrêmement bien adapté aux scripts. Les multiples librairies permettent de traiter efficacement les données et l'aspect haut niveau permet de gérer facilement les appareils. L'écriture est simple et rapide pour l'utilisateur. L'installation du package Python "atmp" depuis PyPi fonctionne parfaitement.

5.3 Améliorations possibles

5.3.1 Interface graphique

1. Améliorer l'apparence de l'interface (plus de couleurs et de séparation entre les éléments)
2. Ajouter des fonctions d'aide à l'utilisateur (adresse IP de l'ordinateur hôte)

5.3.2 Données

1. Utiliser une base de données avec serveur (comme MySQL), ce qui permettrait à l'utilisateur de supprimer les blocs depuis l'interface graphique
2. Intégrer d'autres méthodes de stockage dans les blocs (pas nécessairement en fonction du temps). Cette amélioration est pertinente si une prochaine mise à jour de Grafana met en place la lecture d'oscilloscopes
3. Mettre en place une exportation des blocs dans l'interface graphique ou dans les scripts Python (vers .csv ou excel)

5.3.3 Système

1. Ajouter un service (Windows) qui s'exécute au lancement et qui permettrait de lancer un script au démarrage (monitoring)
2. Compiler le programme pour Linux et le tester sur un Raspberry Pi

5.3.4 Appareils et communication

1. Étudier d'autres possibilités de communication avec les appareils (trigger, actions spécifiques, etc... au lieu de lecture et écriture uniquement)
2. Effectuer des tests de fiabilité sur une longue durée
3. Ajouter un traitement des versions de drivers. Le driver utilisé est sauvegardé et si une nouvelle version arrive, le choix de la version du driver est donné à l'utilisateur. Ceci permet d'éviter qu'un banc de test soit bloqué à cause d'une nouvelle version
4. Ajouter davantage de messages d'erreurs affichés par la librairie Python (type d'erreur, cause, solution possible, etc...)

5.3.5 Ajouts

1. Création d'un circuit low-cost d'entrées/sorties commandable via Modbus TCP ou TCP SCPI (les solutions du marché sont souvent coûteuses). Un tel circuit permettrait d'automatiser facilement des bancs de tests

Chapitre 6

Bibliographie

Programmation

- [6] *Classe pour la gestion des connexions TCP et UDP.* URL : <https://doc.qt.io/qt-5/qabstractsocket.html>.
- [7] *Classe pour la gestion des ports série.* URL : <https://doc.qt.io/qt-5/qserialport.html>.
- [8] *Environnement de développement Qt.* URL : <https://www.qt.io/>.
- [11] *Keysight - VISA shared components.* URL : <https://www.keysight.com/ch/de/lib/software-detail/computer-software/visa-shared-components-downloads-2504667.html>.
- [14] *Librairie Python pour la création de graphs.* URL : <https://matplotlib.org/>.
- [15] *Librairie Python pour le calcul scientifique.* URL : <https://numpy.org/>.
- [16] *Librairie SciPy pour Python.* URL : <https://www.scipy.org/>.
- [23] *Notebook Jupyter.* URL : <https://jupyter.org/>.
- [25] *Outil d'analyse de données en Python.* URL : <https://pypi.org/project/pandas/>.
- [26] *Outil d'installation de packages Python : Pip.* URL : <https://pypi.org/project/pip/>.
- [31] *Signals & Slots.* URL : <https://doc.qt.io/qt-5/signalsandslots.html>.

Ressources

- [2] *Base de données.* URL : https://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es.
- [3] *Base de données MySQL.* URL : <https://www.mysql.com/fr/>.
- [4] *Base de données SQLite.* URL : <https://www.sqlite.org/index.html>.
- [9] *Heure Unix.* URL : https://fr.wikipedia.org/wiki/Heure_Unix.
- [21] *Markdown.* URL : <https://fr.wikipedia.org/wiki/Markdown>.
- [29] *PyPi.* URL : <https://pypi.org/>.
- [30] *SCPI.* URL : https://fr.wikipedia.org/wiki/Standard_Commands_for_Programmable_Instruments.

Outils

- [1] *Agilent IO Librairies Suite.* URL : <https://www.keysight.com/ch/de/lib/software-detail/computer-software/io-libraries-suite-downloads-2175637.html>.
- [19] *Logiciel LabVIEW.* URL : <https://fr.wikipedia.org/wiki/LabVIEW>.
- [24] *Outil d'affichage en fonction du temps.* URL : <https://grafana.com/>.
- [27] *Util de programmation pour STM32.* URL : <https://www.st.com/en/development-tools/stm32cubeprog.html>.

Licences

- [17] *License GNU General Public License.* URL : https://fr.wikipedia.org/wiki/Licence_publique_g%C3%A9n%C3%A9rale_GNU.
- [18] *License GNU Lesser General Public License.* URL : https://fr.wikipedia.org/wiki/Licence_publique_g%C3%A9n%C3%A9rale_limit%C3%A9e_GNU.

Autres

- [10] *IVI Foundation.* URL : <https://www.ivifoundation.org/>.
- [13] *Langage de programmation haut niveau.* URL : https://fr.wikipedia.org/wiki/Langage_de_programmation_de_haut_niveau.
- [20] Marc-Antoine MAILLARD. *Robot testeur de cartes électroniques FUYU FSL40XYZ-L.* URL : <https://tb.heig-vd.ch/7262>.
- [22] *National Instruments.* URL : <https://www.ni.com/fr-ch.html>.
- [32] *Société Betech SA.* URL : <https://www.betech.ch/>.

Table des figures

1.1	Instruments de mesure	8
1.2	Robot	9
1.3	Tête de mesure	9
1.4	Tête de mesure placée sur un circuit	10
1.5	Circuit et "pogo-pins"	10
1.6	Schéma de connexion des appareils	11
2.1	Connecteurs GPIB	15
2.2	Adaptateur USB RS-232	16
2.3	Adaptateur USB - UART (FTDI)	16
2.4	Comparaison des langages	18
2.5	Exemple de script Python avec une alimentation	19
2.6	Exemples de visualisations Grafana	22
3.1	Structure du projet	24
3.2	Résumé des flux de données	25
3.3	Fonctionnement simplifié du programme avec exemple de lecture sur un appareil	26
3.4	Liste des protocoles pour chaque type	27
3.5	Organisation des projets	31
3.6	Écran de démarrage	32
3.7	Nouveau projet	32
3.8	Onglet "Appareils"	33
3.9	Onglet "Scripts"	34
3.10	Onglet "Données"	35
3.11	Onglet "Diagnostic"	36
3.12	Onglet "Paramètres"	37
4.1	Configuration de l'alimentation	41
4.2	Contact entre le circuit et les pogo-pins	42
4.3	Schéma de la mesure de la puissance d'émission	42
4.4	Schéma de la mesure du BER	43
4.5	Schéma de communication entre deux modules	43
4.6	Organisation du script de banc de test	44
4.7	Mesure du BER	49
4.8	Mesure de la puissance	50

Liste des tableaux

1.1	Liste des appareils	11
2.1	Exemples de combinaisons protocole / méthode de communication	14
3.1	Exemple de commande	29
3.2	Exemple de bloc	30
3.3	Emplacement et affichages des types de données	31
3.4	Liste des librairies utilisées	39
4.1	Liste des modulations disponibles	40

Annexe A

Cahier des charges



HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch



Travail de Bachelor
Cahier des charges
Sébastien Deriaz

Département : Technologies industrielles (TIN)
Professeur : Pierre Favrat
Date : 15 mars 2021

Cahier des charges

Table des matières

1	Contexte	1
1.1	Approche	1
1.1.1	Données	2
1.2	Programmation	2
2	Cahier des charges	3
2.1	Généralités	3
2.2	Gestion des appareils	3
2.3	Gestion des données	3
2.4	Scripts	4
2.5	Projets	4
2.6	Autres	4
3	Livrables	4
4	Proposition d'interface	5
5	Signatures	7

Cahier des charges

1 Contexte

Lors de la mise en place d'un banc de test électronique, plusieurs appareils sont utilisés. Ces appareils sont séparés en deux catégories

1. Appareils de laboratoire (oscilloscope, alimentation, multimètre, générateur de signaux etc...). Ces appareils utilisent des protocoles de communication standardisés
2. Appareils "standards" (Microncontrôleurs, Arduinos, CNC etc...). Chaque appareil peut avoir sa méthode de communication spécifique

Pour permettre de coordonner le banc de test, tous les appareils doivent être connectés à un ordinateur qui les contrôle. Ce contrôle se faisant souvent avec un programme comme LabView.

Avantages	Inconvénients
<ol style="list-style-type: none"> 1. Réalisation de bancs de test rapide 2. Communication facilité grâce aux drivers¹ fournis pour les appareils (lorsqu'ils existent) 	<ol style="list-style-type: none"> 1. Connexion à des appareils dont les drivers sont manquants 2. Utilisation d'appareils différents (CNC, Appareils avec communication UART etc...) 3. Opérations arithmétiques complexes (équations, calculs de coordonnées CNC etc...) 4. Coût des licences

1.1 Approche

L'objectif est de réaliser un programme capable de contrôler tous les appareils de laboratoire qui utilisent des protocoles de communication standards. Les appareils standards qui utilisent des moyens de communication similaires mais des syntaxes différentes pourront également être utilisés. Le programme doit permettre de rapidement réaliser des bancs de tests polyvalents avec importation et exportations des données.

Tous les appareils de laboratoire sont compatibles avec, au minimum, un moyen de communication standardisé. Plusieurs méthodes de connexion aux appareils existent mais toutes utilisent la syntaxe du protocole SCPI[1] (dans les cas des appareils de laboratoire). Les méthodes de connexion les plus utilisées sont les suivantes

1. USBTMC[2] : Connexion via un port USB
2. RS-232 : Connexion série (port RS-232 ou émulation via USB)
3. "Raw" TCP/UDP : Connexion par ethernet et transfert des commandes SCPI "brutes" via un canal UDP ou TCP
4. VXI / LXI : Standard le plus récent qui utilise une communication ethernet et le protocole RPC[3].

Les appareils standards utilisent souvent des port série (UART).

La figure 1 montre les différents moyen de communications et la manière dont ils s'organisent dans le modèle OSI.

Cahier des charges

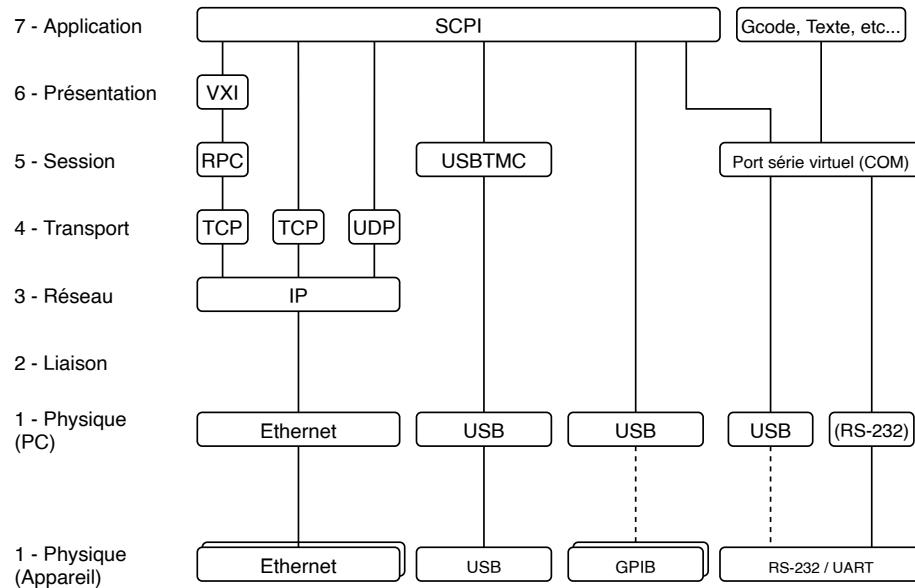


FIGURE 1 – Types de communication

Le protocole SCPI décrit la syntaxe des commandes textuelles permettant de contrôler les appareils.

IDN?	Requête pour obtenir l'identifiant de l'appareil
:Measure:Voltage:DC?	Requête pour obtenir la tension DC mesurée.
:Measure:Current:AC?	Requête pour obtenir le courant AC mesuré.

TABLE 1 – Exemple de commandes SCPI

Pour maximiser la flexibilité des bancs de tests, des scripts python seront utilisés pour décrire les actions et les séquences. Cette méthode permet de tirer parti de toutes les fonctionnalités de python et des différents packages (graphiques, traitement des données, exportations ou importation). Les bancs de tests pourront être sauvegardés avec tous leurs paramètres dans des projets

1.1.1 Données

Les données sont affichées par le programme mais peuvent également être affichées de manière spécifique par les scripts. Les données pourraient également, dans la mesure du possible, être stockées dans des bases de données et être affichées avec des outils comme Grafana²

1.2 Programmation

Pour maximiser la portabilité, la programmation se fera avec Qt³ qui permet de réaliser des applications cross-plateforme en C++. L'environnement Qt met également à disposition des outils pour réaliser des interfaces graphiques.

Le programme sera réalisé et testé sur Windows en priorité. Il est possible que certaines fonctionnalités ne soient pas disponibles sur Linux / macOS à cause de drivers spécifiques (interface USB GPIB Agilent par exemple).

2. <https://grafana.com/>
 3. <https://www.qt.io/>

Cahier des charges

2 Cahier des charges

2.1 Généralités

- 10.10.10 Le programme est développé dans le but d'être cross-plateforme. La priorité est donnée à la plateforme Windows.
- 10.10.20 La compatibilité avec Windows 10 est garantie
- 10.10.30 La langue par défaut de l'interface est le français
- 10.10.40 Une interface en anglais pourrait être mise en place
- 10.10.50 La langue de programmation (variables, commentaires etc...) est l'anglais
- 10.10.60 Dans la mesure du possible, le programme est livré au moyen d'un seul exécutable portable ou d'un seul dossier contenant l'exécutable (pas d'installateur)

2.2 Gestion des appareils

- 10.20.10 La connexion aux appareils possédant un moyen de connexion dans la liste suivante est garantie.
 - 10.20.11 VXI
 - 10.20.12 "Raw" UDP / TCP
 - 10.20.13 Série
 - 10.20.14 UBTMC
- 10.20.20 La connexion aux appareils GPIB sera faite grâce à l'interface Agilent 82357B. La connexion à des appareils GPIB qui n'utilisent pas cette interface n'est pas garantie.
- 10.20.30 Comme l'interface Agilent 82357B utilise des drivers spécifiques d'Agilent ainsi que les outils VISA de National Instruments, la connexion à des appareils GPIB via cette interface n'est pas garantie sur d'autres plateformes que Windows.
- 10.20.40 D'autres protocoles pourraient être implémentés afin d'étendre la polyvalence du programme tel que
 - 10.20.41 ModBus (RTU)
 - 10.20.42 RS-485
- 10.20.50 Le programme dispose d'une liste d'appareil configurable par l'utilisateur. Des appareils peuvent être ajoutés ou supprimés en spécifiant les adresses et paramètres.
- 10.20.60 Des fonctionnalités de tests sont mises en place (test de présence de l'appareil, ping IP, etc...)
- 10.20.70 Pour simplifier la communication avec les appareils par l'utilisateur. Un fichier de correspondance doit être créé pour chaque appareil. Le fichier de correspondance permet de lier des propriétés physiques (tension mesurée, échelle sur un oscilloscope, tension d'alimentation etc...) à une commande.
- 10.20.80 Des fichiers de correspondances génériques seront créés pour traiter les commandes basiques des appareils les plus utilisés (multimètre, oscilloscope etc...)
- 10.20.90 L'utilisateur, par le biais des scripts, peut récupérer ou écrire une valeur d'un appareil de manière simplifiée (Sous réserve que le fonctionnement soit décrit dans le fichier de correspondance).

2.3 Gestion des données

- 10.30.10 Les données des appareils (oscillogrammes, tensions, etc...) sont stockées séparément dans des "blocs". Les blocs peuvent également être créés par les scripts ou générés à partir de données externes (csv, listes etc...)
- 10.30.20 Un affichage rudimentaire des données sera mis en place (graphique, liste, valeurs numériques etc...)

Cahier des charges

- 10.30.30 Un stockage des données dans une base de donnée pourrait être mis en place. Tous les échanges pourraient alors être enregistrés. Les données pourraient également être affichées de manière présentable avec des outils comme Grafana.
- 10.30.40 L'exportation des données peut se faire à partir des scripts python. Il est également possible de sauvegarder les données sous forme de fichiers (.dat, .csv, etc...) et de les récupérer manuellement.

2.4 Scripts

- 10.40.10 Les scripts sont écrits en python
- 10.40.20 Le programme est conçu pour exécuter les scripts à la chaîne. Soit en ligne, soit en boucle pour exécuter des cycles de bancs de test.
- 10.40.30 Un script, via des instructions spécifiques, est capable de configurer entièrement un appareil, de récupérer des données ou de lui en envoyer.
- 10.40.40 Les instructions possibles sont décrites dans le fichier de correspondance et sont affichées par le programme lors de l'écriture des scripts (auto-complétion, volet d'information ou autre).

2.5 Projets

- 10.50.10 Un projet est situé dans un dossier qui porte son nom
- 10.50.20 Le projet peut être enregistré. Tous les paramètres de configuration, la liste des appareils utilisés et les scripts sont sauvegardés au même moment.
- 10.50.30 Il est possible de déplacer un projet sans conséquences (dans le même ordinateur, ou entre ordinateurs)

2.6 Autres

- 10.60.10 Une console est mise en place permettant d'afficher les informations d'exécution des scripts.
- 10.60.20 Un serveur TCP pourrait être mis en place pour permettre de contrôler le programme depuis un autre ordinateur. Chaque ordinateur peut alors servir d'appareil virtuel

3 Livrables

1. Code source C++ avec environnement Qt
2. Exécutable Windows et Linux
3. Rapport
4. Documentation
5. Présentation
6. Projet sur GitHub

Cahier des charges

4 Proposition d'interface

L'interface doit être simple et intuitive. L'écriture des scripts doit être confortable et donc l'espace de l'éditeur important.

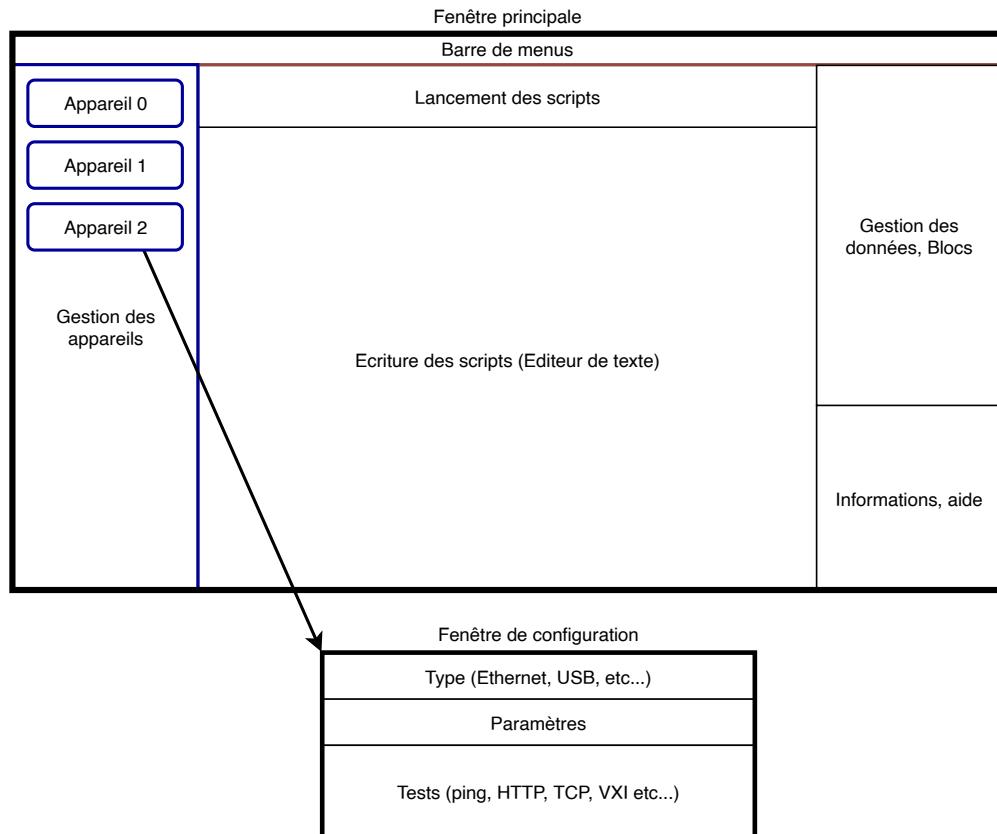


FIGURE 2 – Proposition d'interface graphique

Cahier des charges

Références

- [1] Standard Commands for Programmable Instruments
https://fr.wikipedia.org/wiki/Standard_Commands_for_Programmable_Instruments
- [2] Remote Communication with USBTMC <https://www.testandmeasurementtips.com/remote-communication-with-usbtmc-faq/>
- [3] Remote procedure call https://fr.wikipedia.org/wiki/Remote_procedure_call

Annexes

1. Planning du 15.03.2021

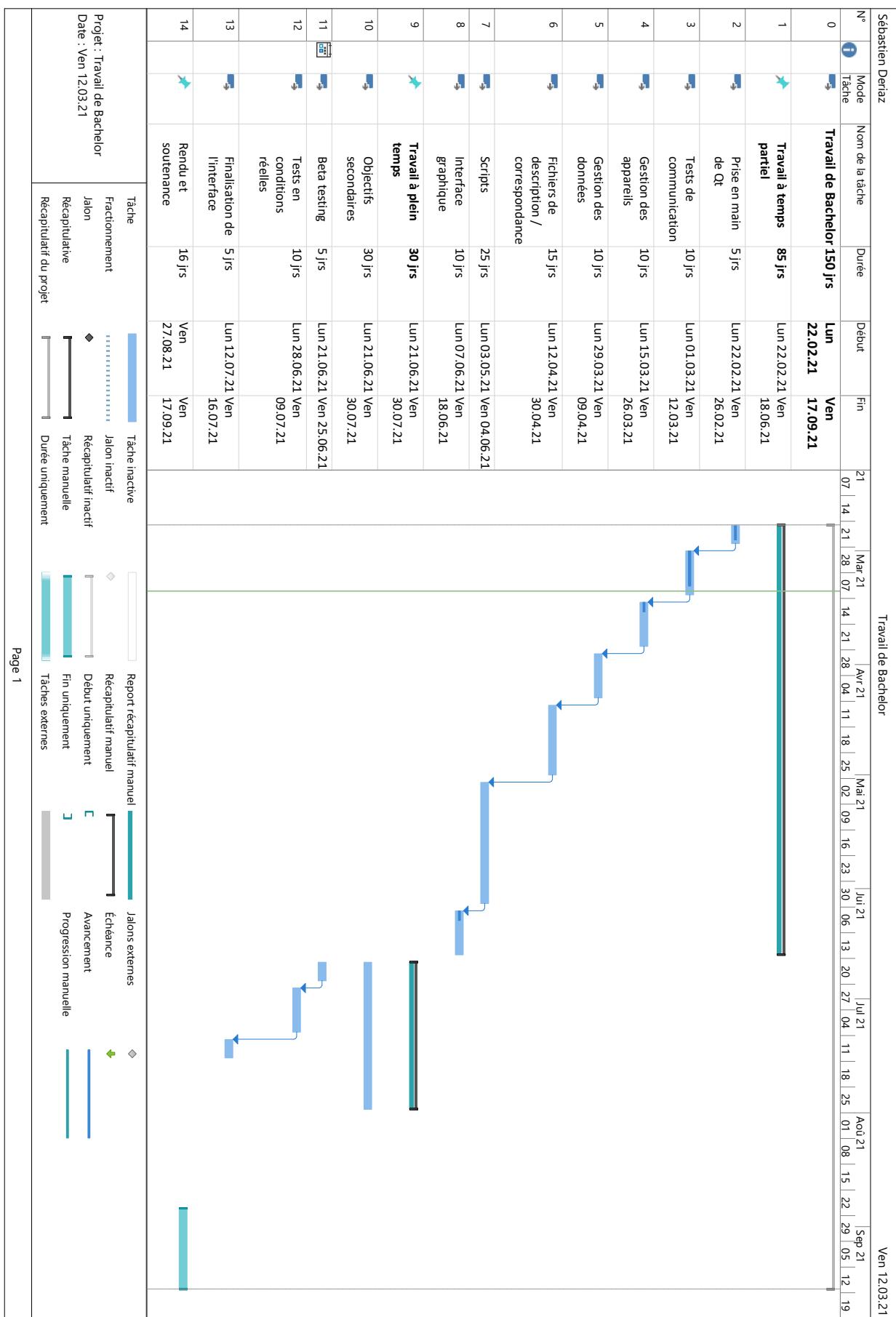
Cahier des charges

5 Signatures

Sébastien Deriaz

Pierre Favrat

Société Betech



Annexe B

Manuel d'utilisation



Département Technologies industrielles (TIN)
Filière Génie électrique
Orientation Électronique embarquée et mécatronique

Travail de Bachelor 2021

Manuel d'utilisation

Étudiant : Sébastien Deriaz
Travail proposé par : Gregory Hänggi
Betech SA
16, Rue du Four
1055 Froideville
Enseignant responsable : Pierre Favrat
Année académique : 2020-2021

Yverdon-les-Bains, le 28 juillet 2021



Table des matières

1	Introduction	3
2	Installation	4
2.1	Interface graphique	4
2.2	Python	5
2.3	Package Python	5
2.4	Visual Studio Code (optionnel)	6
2.4.1	Extensions	7
2.5	Grafana (optionnel)	8
3	Utilisation	9
3.1	Création d'un projet	9
3.2	Appareils	9
3.2.1	Ajout d'un appareil	9
3.2.2	Configuration de l'appareil	9
3.3	Création et lancement d'un script	11
3.3.1	Script	12
3.3.2	Notebook Jupyter	12
3.4	Écriture d'un script	13
3.5	Driver	13
3.6	Données	14
3.7	Console	14
4	Bibliographie	15



Chapitre 1

Introduction

Ce document constitue le manuel d'utilisation de la plateforme ATMP¹.

La plateforme est constituée de deux parties

1. Interface graphique
2. Package Python

L'installation sera décrite pour les deux parties.

Les procédures présentées dans ce document sont destinées à un système Windows.

1. Automated Test & Measurement Platform



Chapitre 2

Installation

2.1 Interface graphique

La version actuelle du logiciel est livrée sous forme portable dans une archive .zip. Les étapes pour l'installation du programme sont

1. Téléchargement de l'archive .zip[3]
2. Décompression de l'archive
3. Déplacement du dossier à l'emplacement choisi¹

Le programme peut être démarré en double cliquant sur l'exécutable **ATMP.exe**.

Un raccourci peut également être créé en effectuant un clic-droit sur l'exécutable puis "Créer un raccourci". Il est aussi possible de glisser l'exécutable sur le bureau tout en maintenant CTRL+SHIFT avant de le relâcher. Le raccourci sera créé directement sur le bureau.

1. Typiquement Program Files pour Windows

2.2 Python

Pour installer Python (sur Windows²), il est nécessaire de suivre les instructions suivantes :

1. Téléchargement de l'installateur[6]
2. Lancement de l'installateur

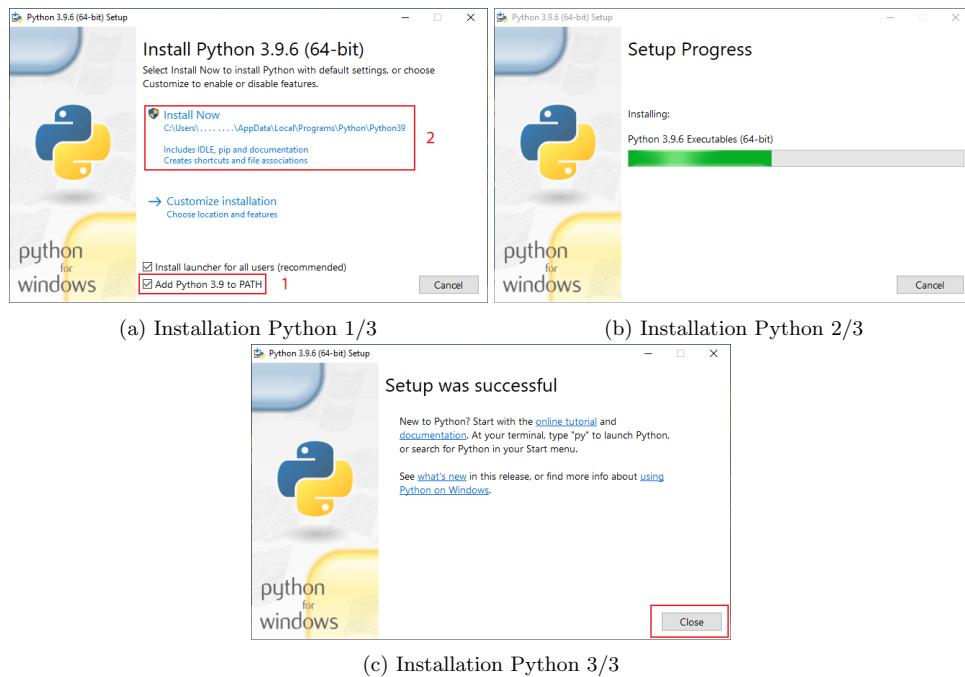


FIGURE 2.1 – Installation de Python

L'option "Add Python 3.9 to PATH" permet d'appeler une ligne de commande Python. Pour vérifier si l'installation a été faite correctement, il est possible d'ouvrir un terminal et de taper

```
python --version
```

Si la ligne de commande retourne "Python 3.x.x", alors l'installation a été effectuée avec succès. Sinon, il faut recommencer à partir de l'étape 1).

2.3 Package Python

Le package pyatmp[4] est installé grâce à l'outil *pip*[2]. *Pip* est installé en même temps que Python. Si ce n'est pas le cas, il est possible de l'installer manuellement.

Le package[5] est disponible sur PyPi³. Pour l'installer, il suffit de lancer la commande

```
pip install pyatmp
```

2. Il est également possible d'utiliser Python sur WSL mais cette méthode ne sera pas traitée ici
 3. Python Package Index : pypi.org

2.4 Visual Studio Code (optionnel)

Visual studio code permet d'éditer les scripts ainsi que les notebooks Jupyter. Il est fortement recommandé de l'installer. L'installation se fait de la manière suivante :

1. Téléchargement de l'installateur[7]
2. Lancement de l'installateur

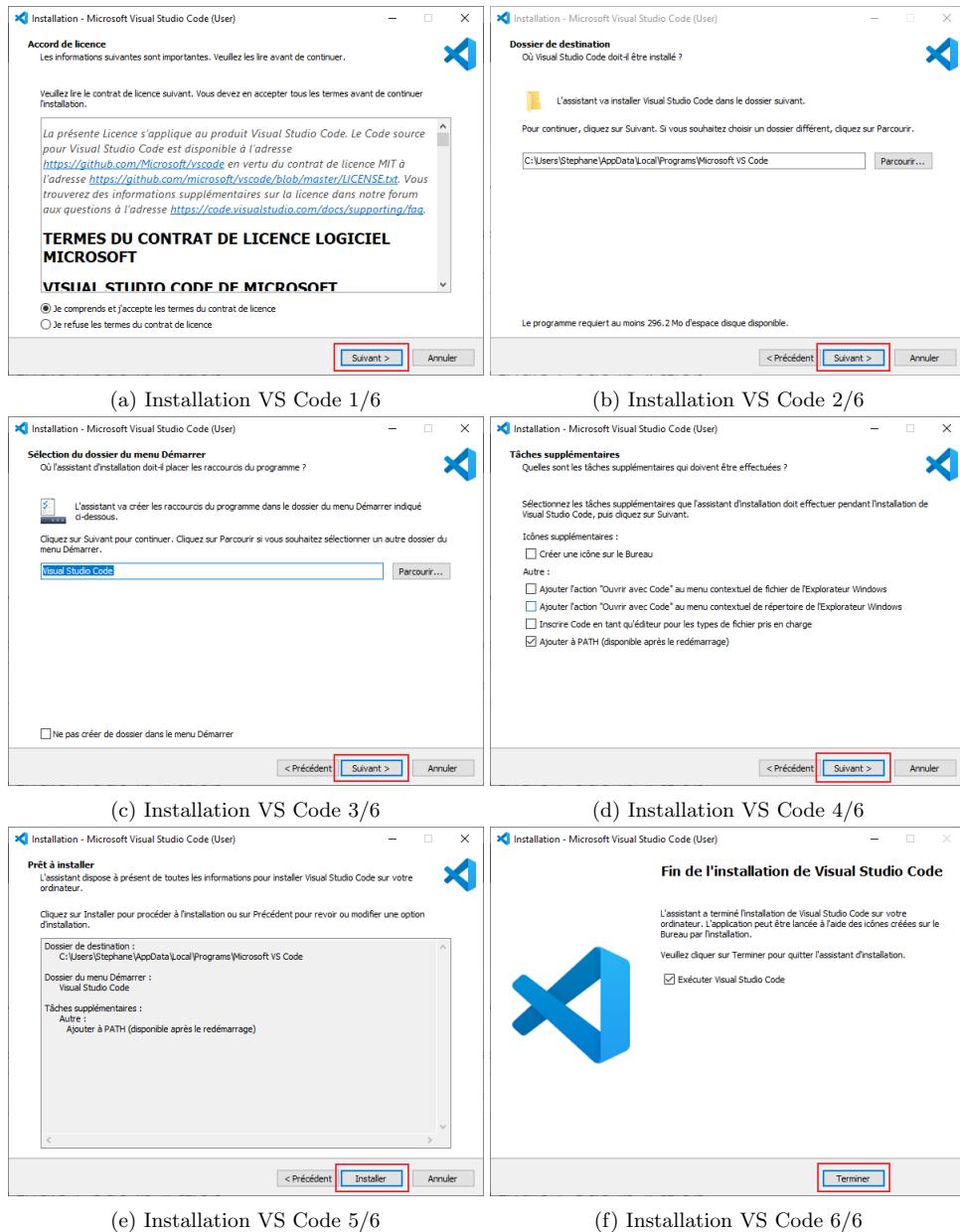


FIGURE 2.2 – Installation de Visual Studio Code

2.4.1 Extensions

Si elles ne sont pas déjà installées, les extensions suivantes doivent être ajoutées

- Python
- Jupyter

Pour installer les extensions il faut aller dans l'onglet "Extensions" sur la gauche puis taper le nom dans la barre de recherche.



FIGURE 2.3 – Onglet "Extensions"

2.5 Grafana (optionnel)

Grafana est utilisé pour visualiser les données créées lors de l'exécution des scripts.

Pour l'installer il est nécessaire de suivre les instructions suivantes :

1. Téléchargement de l'installateur[1] (version "self-managed")
2. Lancement de l'installateur

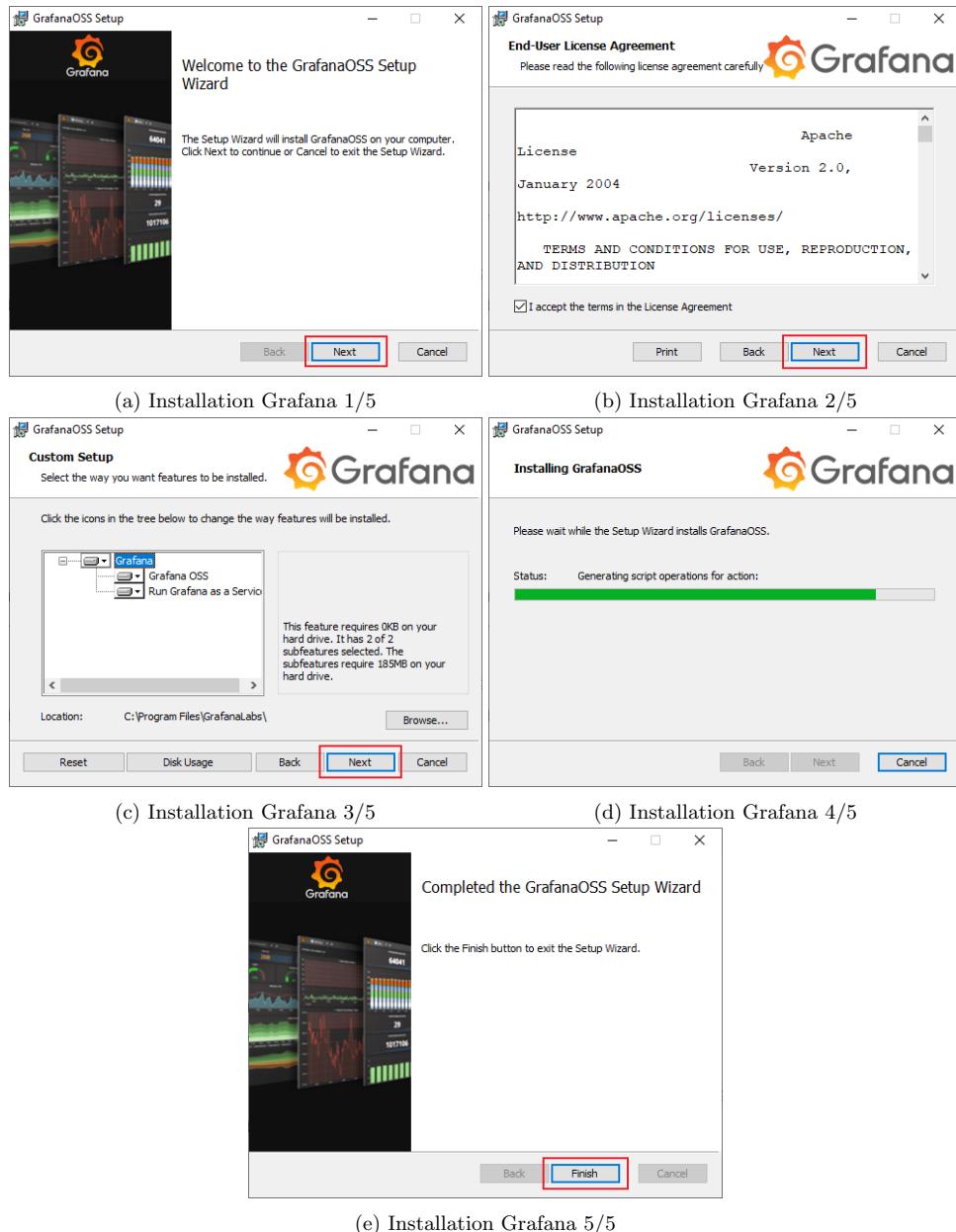


FIGURE 2.4 – Installation de Grafana

Chapitre 3

Utilisation

3.1 Crédation d'un projet

Lors du premier démarrage, un projet doit être créé.

3.2 Appareils

3.2.1 Ajout d'un appareil

Pour ajouter un appareil, il suffit de cliquer sur le bouton "+" au dessus de la liste (voir figure 3.5).

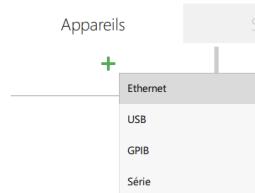


FIGURE 3.1 – Ajout d'un appareil

Le type d'appareil doit ensuite être choisi (Ethernet, USB, GPIB ou Série).

3.2.2 Configuration de l'appareil

Chaque appareil possède

1. Un nom : description courte de l'appareil (les caractères spéciaux sont autorisés)
2. Un alias : un nom court utilisé pour référencer l'appareil (seules les minuscules, majuscules et chiffres sont autorisés)
3. Un driver : fichier .py qui permet de lier les commandes Python à la syntaxe spécifique pour chaque appareil

Le protocole permet de modifier la manière dont la communication est effectuée. Cette option est utilisée principalement pour les appareils de type Ethernet.



Ethernet

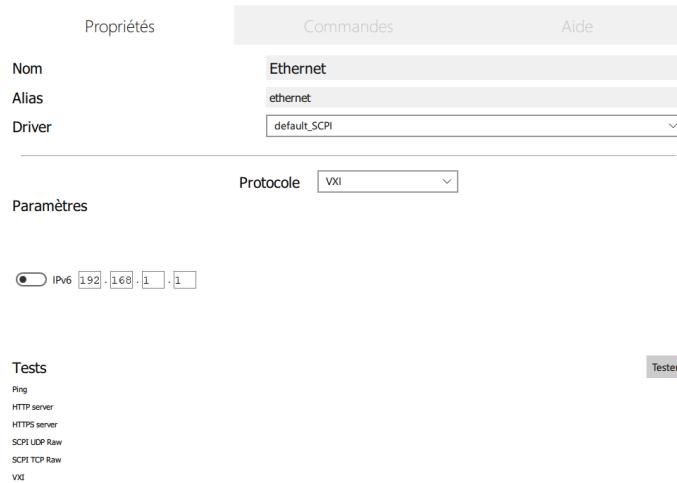


FIGURE 3.2 – Propriétés Ethernet

Les appareils Ethernet nécessitent uniquement une adresse IP. Il est possible de configurer une adresse IPv4 ou IPv6

USB

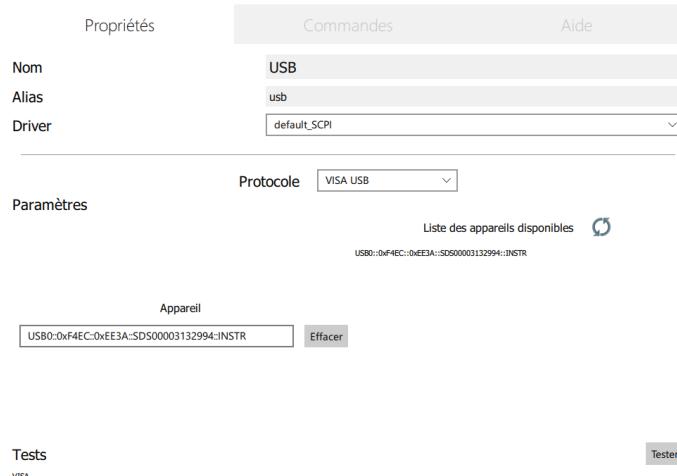


FIGURE 3.3 – Propriétés USB

Les appareils USB (compatibles VISA) sont affichés dans une liste d'appareils, sur la droite. Pour sélectionner un appareil, il suffit de faire un double-clic sur son nom.



GPIB

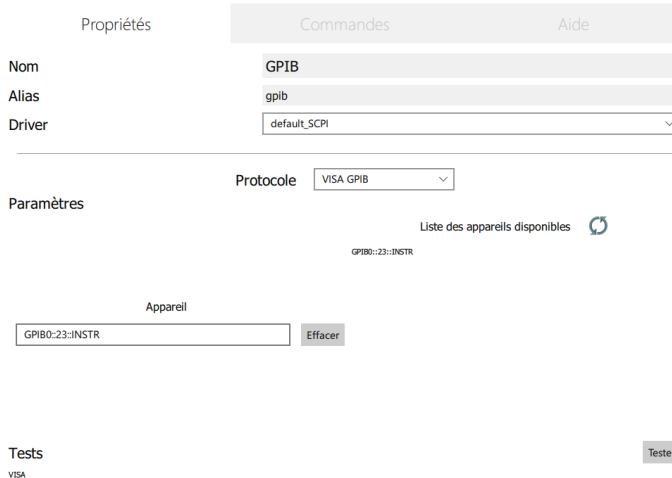


FIGURE 3.4 – Propriétés GPIB

Comme pour les appareils USB, les appareils GPIB sont affichés dans une liste et le choix s'effectue avec un double-clic.

Série

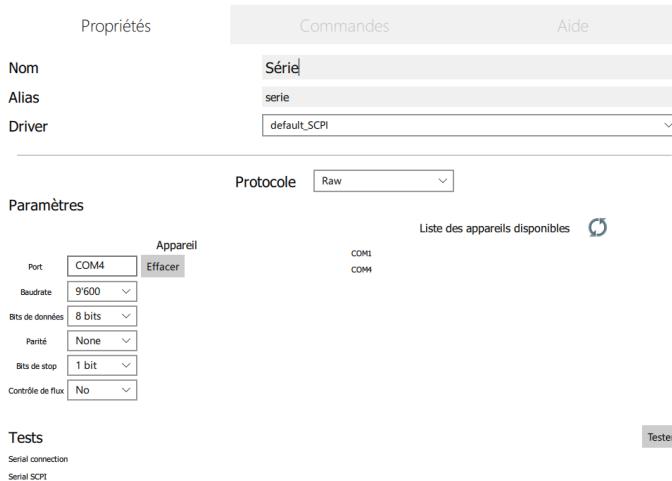


FIGURE 3.5 – Propriétés série

Le choix du port série s'effectue avec un double-clic dans la liste sur la droite.
La communication doit également être configurée (baudrate, bits de données, etc...)

3.3 Crédation et lancement d'un script

Il existe deux possibilités pour exécuter du code Python

1. Script
2. Notebook Jupyter

3.3.1 Script

Un script Python est un fichier .py qui peut être exécuté depuis l'interface graphique (voir 3.6) ou par ligne de commande

```
python <nom_script>
```



FIGURE 3.6 – Lancement d'un script depuis l'interface

Un script peut être modifié à l'aide d'un simple éditeur de texte comme Notepad++.

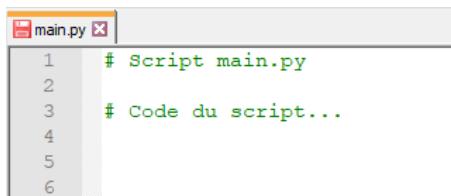


FIGURE 3.7 – Fichier de script

3.3.2 Notebook Jupyter

Un notebook peut être créé avec JupyterLab ou Visual Studio Code. Ce manuel utilisera Visual Studio Code, car il est plus facile de l'installer sur Windows.

Pour créer un notebook sur Visual Studio Code il faut suivre les étapes suivantes :

1. Ouvrir Visual Studio Code
2. Fichier > Nouveau Fichier (ou CTRL+N)
3. Sélectionner le langage "Jupyter"
4. Enregistrer le fichier. Il est important de garder l'extension lors de l'enregistrement ".ipynb"
5. Fermer le fichier pour le réouvrir¹

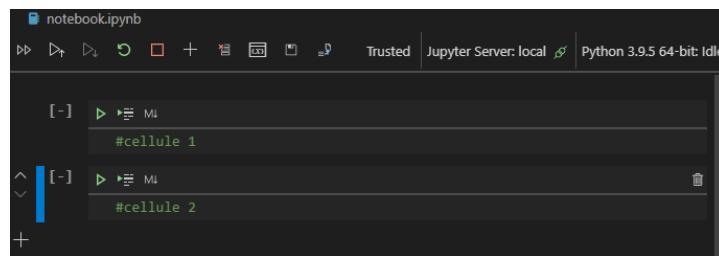


FIGURE 3.8 – Exemple de notebook

Le code est organisé en cellules qui peuvent être exécutées les unes après les autres avec CTRL+SHIFT. Les notebooks Jupyter ne peuvent pas être lancés depuis l'interface.

¹. Ceci est nécessaire pour actualiser l'interface du notebook

3.4 Écriture d'un script

La procédure est la même pour l'écriture d'un script Python ou un notebook Jupyter.

```

1 ## Exemple de script
2 # importation de librairies Python
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 # importation de la librairie de communication
7 import atmp
8
9 # Déclaration d'un appareil (avec l'alias "mult")
10 multimeter = atmp.devices.device('mult')
11
12 # Mesure de tension
13 tension = multimeter.getDCVoltageMeasurement()

```

Pour qu'un appareil soit utilisable dans le script il faut que :

1. Il soit déclaré dans l'interface graphique avec un alias
2. Un driver soit sélectionné

Les commandes (`multimeter.xxx`) sont les fonctions déclarées dans le driver.

3.5 Driver

Un driver est obligatoire pour communiquer avec un appareil. Un driver est de la forme

```

1 # Création de la classe (ne doit pas être modifié)
2 class new():
3     # Initialisation de la classe (ne doit pas être modifié)
4     def __init__(self, device):
5         self.device = device
6
7     # Fonctions de base (écriture / lecture de données brutes)
8     def writeRaw(self, data : bytearray):
9         self.device.write(data)
10
11    def readRaw(self):
12        return self.device.read()
13
14    # Création de fonctions spécifiques
15    def getDCVoltageMeasurement(self):
16        self.device.write('MEAS:VOLT:DC?\n')
17        return float(self.device.read())

```

3.6 Données

Les oscilloscopes, captures d'écran, etc... sont stockés dans des fichiers spécifiques (.png, .csv, .dat) par Python. La liberté est laissée à l'utilisateur.

Le stockage de données temporelles "lentes" (avec une période inférieure à la seconde) peut être effectué dans des blocs.

```

1 # Importation de la librairie de communication
2 import atmp
3
4 # Déclaration d'un bloc
5 progression = atmp.blocks.block('progression')
6
7 # Nettoyage du bloc
8 progression.clear()
9
10 # Enregistrement d'une valeur unique
11 progression.log(4)
12
13 # Enregistrement de plusieurs valeurs
14 progression.log([1,2,3])

```

3.7 Console

La console est un bloc spécial utilisé pour stocker des informations sous forme de texte avec un niveau de gravité d'événement.

```

1 # Importation de la librairie de communication
2 from atmp import console
3
4 # Nettoyage de la console
5 console.logInfo("Ceci est un message de niveau info")
6 console.logError("Ceci est un message de niveau erreur")
7 console.logCritical("Ceci est un message de niveau critique")

```

Les niveaux disponibles sont

1. info (logInfo)
2. trace (logTrace)
3. debug (logDebug)
4. warning (logWarning)
5. error (logError)
6. critical (logCritical)
7. unknown (logUnknown)

Les messages sont ensuite affichés dans l'interface graphique (Scripts > Console client)

```

Ceci est un message de niveau info
Ceci est un message de niveau erreur
Ceci est un message de niveau critique

```

FIGURE 3.9 – Affichage de la console dans l'interface graphique

Chapitre 4

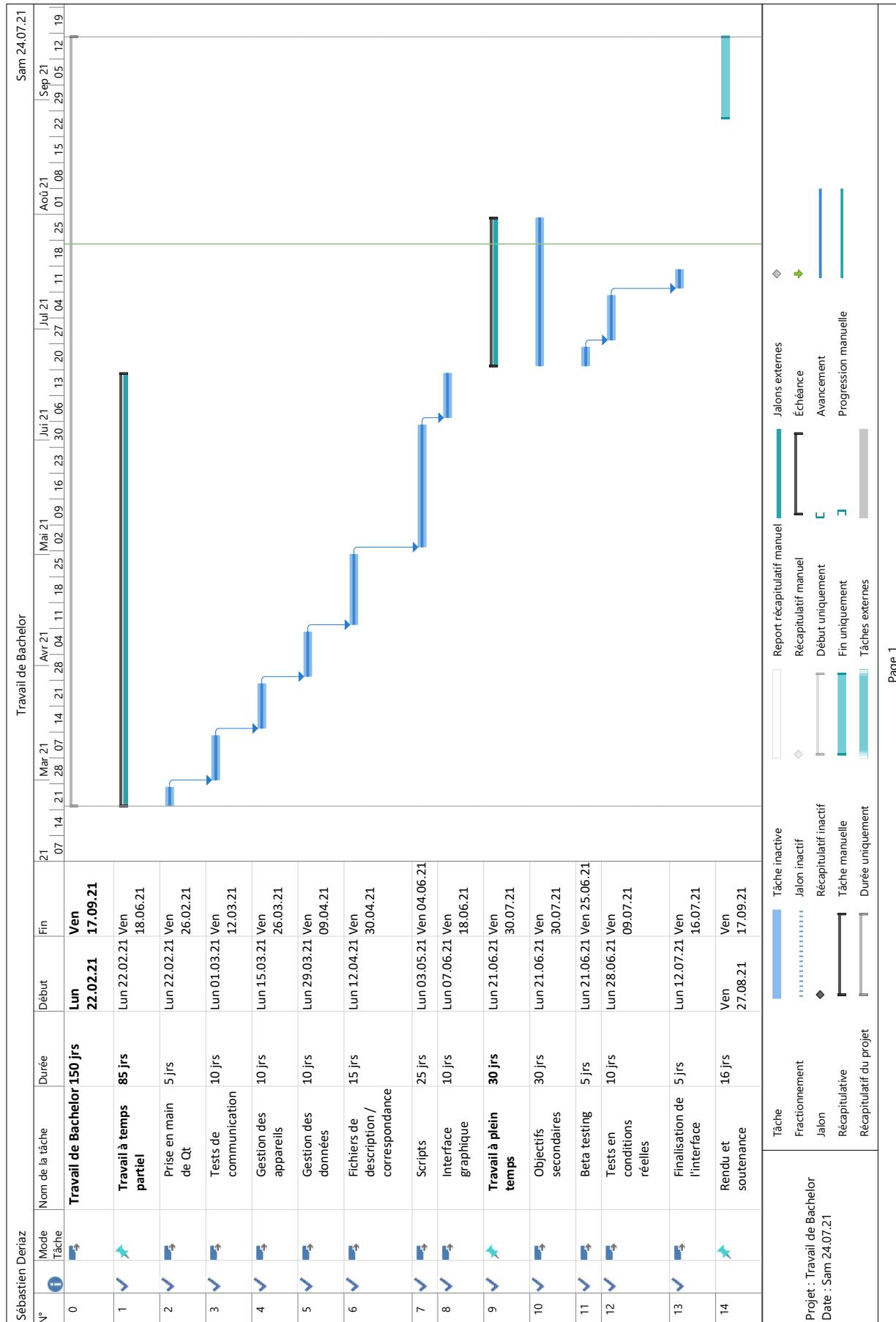
Bibliographie

Liens

- [1] *Installateur pour Grafana.*
URL : <https://grafana.com/grafana/download?pg=get&plcmt=selfmanaged-box1-cta1&platform=windows>.
- [2] *pip.*
URL : <https://pypi.org/project/pip/>.
- [3] *Projet ATMP (Interface graphique).*
URL : <https://github.com/SebastienDeriaz/ATMP>.
- [4] *Projet pyatmp (Package python).*
URL : <https://github.com/SebastienDeriaz/pyatmp>.
- [5] *projet pyatmp sur PyPi.*
URL : <https://pypi.org/project/pyatmp/>.
- [6] *Téléchargement de Python.*
URL : <https://www.python.org/downloads/>.
- [7] *Visual Studio Code.*
URL : <https://code.visualstudio.com/>.

Annexe C

Planning (24.07.2021)



Annexe D

Licence VISA Shared Components

Ce document est affiché lors de l'installation de VISA Shared Components x64, téléchargé depuis le site de Keysight[11]

LICENSE AGREEMENT

BEFORE YOU CLICK ON THE ACCEPT BUTTON AT THE END OF THIS DOCUMENT, CAREFULLY READ ALL THE TERMS AND CONDITIONS OF THIS AGREEMENT. BY CLICKING ON THE ACCEPT BUTTON, YOU ARE CONSENTING TO BE BOUND BY AND ARE BECOMING A PARTY TO THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL OF THE TERMS OF THIS AGREEMENT, CLICK THE "DO NOT ACCEPT" BUTTON AND DO NOT DOWNLOAD AND/OR USE THIS INTELLECTUAL PROPERTY.

Readers of this document are requested to submit to Interchangeable Virtual Instruments, Inc. (« Licenser »), with their comments, notification of any relevant patent rights or other intellectual property rights of which they may be aware which might be infringed by any use of this intellectual property, software, or specification (the « Intellectual Property »), as appropriate, and to provide supporting documentation.

Copyright © 2002, Interchangeable Virtual Instruments Foundation, Inc. All Rights Reserved. IVI Foundation is the exclusive licensee of the « IVI » trademark and the Interchangeable Virtual Instruments Foundation, Inc. logo.

Attention is drawn to the possibility that some of the elements of this Intellectual Property may be the subject of patent or other intellectual property right (collectively, "IPR") of third parties. LICENSOR shall not be responsible now or in the future for identifying any or all such IPR.

Permission is hereby granted, free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that the above copyright notice(s) appear in all copies of the Intellectual Property and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement. If you are not a member of LICENSOR, your license hereunder is limited to the use of the object code of the Intellectual Property. If you are a member of LICENSOR, your license extends to the source code of the Intellectual Property.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

You may not charge for any sublicense of the Intellectual Property; provided however, that the Intellectual Property may be sublicensed together with another product so long as there is no separate charge for the Intellectual Property.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF IPR TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. It will also terminate if you fail

to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of the copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the State of Delaware. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

The Intellectual Property is a "commercial item," as that term is defined in 48 C.F.R. 12.101 (Oct. 1995), consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire the Intellectual Property with only those rights set forth herein.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.