Hes·so VALAIS WALLIS

School of Engineering

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Filière

Énergie et technique environnementale

Orientation Smart Grid

TRAVAIL DE BACHELOR

DIPLÔME 2024

Alexandre Lê-Agopyan

Realization of a phasor data concentrator for phasor measurement units

Professor
Dr. Fabrizio Sossan, fabrizio.sossan@hevs.ch

Expert
Dr. Lorenzo Zanni, lorenzo.zanni@zaphiro.ch

Date de la remise du rapport
16 August 2024

Valais excellence CERTIFIED

Données du travail de bachelor
Aufgabenstellung der Bachelorarbeit

Filière / Studiengang **ETE**	Année académique / *Studienjahr* **2023-24**	No TB / *Nr. BA* **SG/2024/65**
Mandant / *Auftraggeber* ☒ HES—SO Valais-Wallis ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*	Etudiant·e / *Student/in* **Alexandre Lê-Agopyan** Professeur·e / *Dozent/in* **Fabrizio Sossan**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais-Wallis ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja ☒ non / nein	Expert·e / *Experte/Expertin* *(données complètes/vollständige Angaben)*	

Titre / *Titel*

Realization of a phasor data concentrator for phasor measurement units

Description / *Beschreibung*

The objective of the work is to support the development of the installation of phasor measurement technology in the Gridlab. The student will have the pivotal task of developing the code to access the measurements from phasor measurement unit (PMU), store it in a time series database, and visualize it. In addition, the student will lead the installation and configuration of the system to synchronize PMUs in the Gridlab. This work will be demonstrated and validated by developing a monitoring system to visualize PMU measurements from one of the Gridlab's test benches (i.e., the "dispatch").

Objectifs / *Ziele*

- Development of a demo code that establishes connection with a PMU and extracts measurements.
- Setup of the CiscoSystem system to synchronize multiple PMUs.
- Extension of the software to n PMUS and design of a configuration file to enable the definition of available measurements.
- Saving of the data to an influxdb database.
- Creation of a dashboard to visualize measurements applied to the dispatch case study.

Signature ou visa / *Unterschrift oder Visum*	Délais / *Termine*
Responsable de l'orientation / *Leiter/in der Vertiefungsrichtung:* ..... [1] Etudiant·e / Student/in:	Attribution du thème / *Ausgabe des Auftrags:* **13.05.2024** Présentation intermédiaire / *Zwischenpräsentation*: **Semaine / Woche 25** (17-21.06.2024) Remise du rapport final / *Abgabe des Schlussberichts:* **16.08.2024, 12:00** Expositions / *Ausstellungen der Diplomarbeiten:* **23.08.2024** – HEI **26.08.2024** – Monthey **29.08.2024** – Visp Défense orale / HEI *Mündliche Verfechtung:* **Semaine / Woche 35-36** (29.08-05.09.2024)

[1] *Par sa signature, l'étudiant·e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de bachelor.*
Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.

Hes·so VALAIS WALLIS

Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

Realization of a phasor data concentrator for phasor measurement units

Diplômant/e Alexandre Lê-Agopyan

Objectif du projet

Ce projet consiste à développer un système de mesure de phaseurs de tension et de courant sur le dispatch du Gridlab. Le système de mesure permet d'analyser précisément et en temps réel les phaseurs de courant et de tension du réseau afin d'analyser son état. Les informations collectées par les appareils de mesure ainsi que l'état du réseau sont envoyés sur une base de données afin d'être consultable par l'opérateur du système.

Méthodes | Expériences | Résultats

Les appareils de mesure sont installés sur le modèle de réseau de dispatch présent au Grid Lab. La stratégie d'implémentation des appareils de mesure est effectuée au préalable avec l'équipe du laboratoire.
Les Phasor Measurement Units (PMUs) peuvent mesurer 3 phaseurs de tension ainsi que 15 courants. Les données collectées par les PMUs sont encodées à l'aide de la norme IEEE C37.118. Les données des appareils nécessitent d'être décodées afin de pouvoir être utilisées correctement. Le décodage de ces données est effectué à l'aide d'un programme Python réalisé dans ce but.
L'intérêt d'utiliser des PMUs réside dans la synchronisation des appareils entre eux. Plusieurs appareils sont connectés sur le même réseau et nécessitent une synchronisation temporelle mutuelle. Les appareils sont ainsi connectés au moyen d'une base de temps commune donnée par un clock de référence externe.
Un système de mesure de phaseurs sur le dispatch a été réalisé. Les appareils de mesure ont été placés et paramétrés. Les données mesurées sur le dispatch sont envoyées sur une base de données permettant leur visualisation. Des incohérences lors de la mesure des courants empêchent le bon fonctionnement d'algorithmes de calcul d'impédance et de validation des admittances.

Travail de diplôme
| édition 2024 |

Filière
Energie et techniques environnementales

Domaine d'application
Smart Grid

Professeur responsable
Sossan Fabrizio
fabrizio.sossan@hevs.ch

Information about this report

Contact Information

Author:	Alexandre Lê-Agopyan
	Bachelor Student
	HEI-Vs
Email:	alexandre.le-agopyan@hevs.ch

Declaration of honor I, undersigned, Alexandre Lê-Agopyan, hereby declare that the work submitted is the result of a personal work. I certify that I have not resorted to plagiarism or other forms of fraud. All sources of information used and the author quotes were clearly mentioned.

Place, date: Sion, 16.08.2024

Signature: 

Remerciements

Je tiens à exprimer toute ma gratitude envers les personnes qui ont grandement contribué à la réussite de ce projet.

Tout d'abord, je souhaite sincèrement remercier Fabrizio Sossan pour m'avoir offert l'opportunité de travailler sur ce sujet, ainsi que pour ses précieux conseils et son accompagnement tout au long du projet.

Mes remerciements vont également à Alain Roduit pour son aide concernant l'installation des sondes de courant, les mesures de tension, l'alimentation des appareils et la conception du chariot de support.

Enfin, je tiens à exprimer ma reconnaissance envers Didier Blatter pour son soutien et son expertise lors de la résolution des problèmes techniques.

Abstract

Les énergies renouvelables se développent massivement et nécessitent une attention particulière. Le système actuel de transport et de distribution est vieillissant et n'a pas été initialement prévu pour supporter de tels pics de puissance. Les apports énergétiques sont majoritairement issus des panneaux photovoltaïques et sont difficilement prévisibles. La solution des Phasor Measurement Units (PMUs) vient ajouter une supervision supplémentaire de l'état du réseau. L'ajout de tels dispositifs de mesure permet de mieux visualiser le comportement des points d'injection du réseau de distribution et de s'y adapter en conséquence.
Utilisés traditionnellement dans les lignes de transmission, les PMUs permettent de mesurer les phaseurs des tensions nodales et des courants de ligne. L'analyse des phaseurs à différents nœuds requiert une certaine synchronisation des mesures. Les PMUs utilisent une base de temps commune généralement fournie par GPS. Les PMUs permettent d'implémenter des algorithmes d'estimation d'état linéaire et de détection de défaut grâce à leur taux d'acquisition élevé et leurs phaseurs de courant et de tension. Les smarts meters utilisés dans les réseaux de distribution ne permettent pas de réaliser ce genre de tâche.

Dans le cadre de ce travail, les PMUs sont installés sur un modèle de réseau de dispatch développé à des fins pédagogiques et de recherche. Cette thèse décrit les procédures d'installation de PMUs dans le gridlab, leur configuration, et le développement du Phasor Data Concentrator (PDC) permettant de récupérer les mesures des PMUs afin de les enregistrer dans une base de données temporelles.
La thèse décrit également deux applications créées dans le but de de valider et d'utiliser les données des PMUs. La première consiste à calculer les impédances des lignes (qui devraient être similaires avec les valeurs physiques). La seconde application fait référence à la validation de la matrice d'impédance du bus du système Y, suivant la relation I = YV. Finalement, la thèse décrit un problème rencontré avec l'acquisition des mesures ou les phaseurs de courant de lignes, devant normalement être constants, montraient des comportements dynamiques. Plusieurs tests et hypothèses ont été réalisés afin d'identifier la source du problème. Des analyses supplémentaires sont nécessaires pour trouver la raison qui pousse les mesures à adopter un comportement dynamique.

Mots clés: *Wide Area Monitoring, Phasor Measurement Unit, Smart Grid, Monitoring, Réseau*

Durabilité

Les objectifs de développement durable (ODD) constituent un appel universel à l'action pour mettre fin à la pauvreté, protéger la planète et améliorer la vie et les perspectives de chacun, partout dans le monde. [1] Le développement d'un système de mesure sur un réseau de distribution se positionne sur les objectifs sept et neuf établis par l'organisantion des nations unies (ONU).

L'objectif sept a comme titre: énergie propre et d'un coût abordable. Son but est de garantir l'accès de tous à des services énergétiques fiables, durables et modernes à un coût abordable. [1]

L'objectif neuf a comme titre: industrie, innovation et infrastructure. Le but est de développer le développement durable en investissant dans des infrastructures telles que les transports, l'irrigation, l'énergie et les technonologies de l'information et de la communication. [1]

L'implémentation de systèmes de mesure intelligents sur le réseau contribue à l'objectif neuf en renforçant la structure déjà existante. Renforcer le réseau vient le fiabiliser et facilite le développement de celui-ci ce qui se positionne sur l'objectif sept.

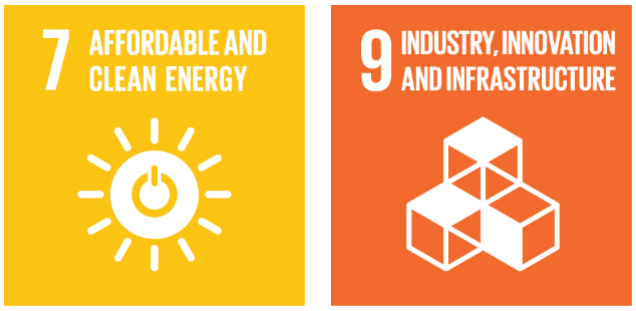

Fig. 1. – Objectifs de développement durable 7 et 9

Table des matières

Figures

Tableaux

Équations

I Introduction

Table des matières

1 Problématique

L'intégration des nouvelles sources énergétiques incluant les énergies renouvelables, les véhicules électriques ont considérablement modifié les profils de charge du réseau. Ces ajouts rendent la supervision et le contrôle des réseaux de distribution compliqué. À l'heure actuelle la plupart des gestionnaires de réseau utilisent le système Supervisory Control And Data Acquisition (SCADA) pour monitorer et contrôler le réseau. Le système SCADA récupère des données de tension, de courant, de puissance active et réactive de manière non-synchronisée avec une faible résolution. Le système SCADA ne peut donc pas récupérer le comportement dynamique du réseau de distribution et rend son monitoring compliqué. [5]

2 Objectif

L'objectif de ce travail est d'apporter un système de mesure permettant de résoudre les problèmes de supervision rencontrés avec le système SCADA. Le système SCADA doit être remplacé par un système plus précis et synchronisé dans le temps.

Le choix des appareils de mesure s'est porté pour ce projet sur les Phasor Measurement Units (PMUs) qui ont la possibilité de communiquer entre eux de manière synchronisée. Le décryptage des données des appareils ainsi que la communication entre les appareils de mesure doivent être établies. Un algorithme de détection d'état du réseau doit également être réalisé.

3 Structure du rapport

3.1 État de l'art

L'état de l'art vient éclaircir l'intérêt des PMUs et des PDCs sur le réseau existant. Cette étape du projet apporte beaucoup d'informations sur les méthodes déjà utilisées sur la technologie de mesure ainsi que de ses différents avantages et inconvénients.

3.2 Câblage et configuration

Ce chapitre explique les différentes étapes nécessaires à la construction d'un réseau de mesures à l'aide des PMUs et du Switch. Le câblage et l'emplacement des appareils y sont décrits. Les différentes étapes de configuration du Switch et des PMUs sont également indiquées dans le cas où des paramètres viendraient à être modifiés à l'avenir.

3.3 Développement

Le chapitre développement décrit la structure de base du travail sur lequel les différentes applications annexes viennent puiser les informations. Cela comprend l'acquisition des données, leur traitement et leur envoi sur la base de données. Les séquences de démar-

rage du système de mesure y sont également décrites. Le chapitre contient également une description de certaines méthodes et protocoles importants pour le bon fonctionnement du système.

3.4 Applications

Dans ce chapitre, les différentes applications utilisant les données envoyées par les PMUs sont décrites. Ces applications récupèrent et utilisent les données stockées sur la base de données. Le développement des différentes applications et leur fonctionnement sont expliqués.

3.5 Implémentation sur Raspberry Pi

Durant la phase initiale, tous les codes étaient employés sur un ordinateur portable. L'ensemble du projet a été transféré sur un Raspberry Pi afin que celui-ci puisse tourner de manière autonome en continu. Les différentes procédures permettant d'utiliser le Raspberry Pi sont décrites dans ce chapitre.

3.6 Problèmes rencontrés

Ce chapitre recense tous les problèmes majeurs rencontrés lors du développement du projet. Les protocoles de tests réalisés ainsi que les différentes solutions trouvées y sont indiquées.

3.7 Conclusion

La conclusion contient un résumé du projet, une comparaison avec les objectifs initiaux, les problèmes rencontrés ainsi que les perspectives d'amélioration futures.

3.8 Documentation du software

La documentation du software développé est présent dans les annexes. Ce document indique l'emplacement dans le rapport des informations liées à l'utilisation du code réalisé. (voir Annexe A)

II État de l'art

Table des matières

1 Objectif de l'état de l'art

L'objectif de l'état de l'art est de passer en revue la littérature liée à la thématique de ce travail. Il est important d'analyser l'avancement actuel de la technologie afin de permettre une compréhension plus précise du sujet.

2 Méthodologie

Pour traiter ce sujet, différents points sont abordés. Ce chapitre passe en revue les points clés du projet en commençant par une description globale et l'utilité des Phasor Measurement Units sur le réseau. Une description globale du rôle d'un Phasor Data Concentrator sera également effectuée. Les aspects plus techniques, incluant les solutions actuelles, les problématiques ainsi que les méthodes, seront ensuite analysés.

3 État actuel

3.1 Phasor Measurement Units

Les PMUs permettent de mesurer des signaux sinusoïdaux qui sont l'élément principal des circuits ACs. Les phaseurs sont les principaux composants et permettent de visualiser les variations de courant et de tension dans le temps. Les performances du réseau peuvent être analysées à l'aide de phaseurs. Pour l'analyse du réseau électrique, les PMUs peuvent être séparés par des distances importantes. Afin d'assurer la synchronisation de toutes les mesures, les appareils sont soumis à la même constante de temps établie par Global Positioning System (GPS).

L'estimation d'état du réseau est une des applications possibles avec les PMUs. En récupérant plusieurs mesures d'appareils disposés à plusieurs endroits sur le réseau, il est possible d'après sa topologie de définir son état. [2]

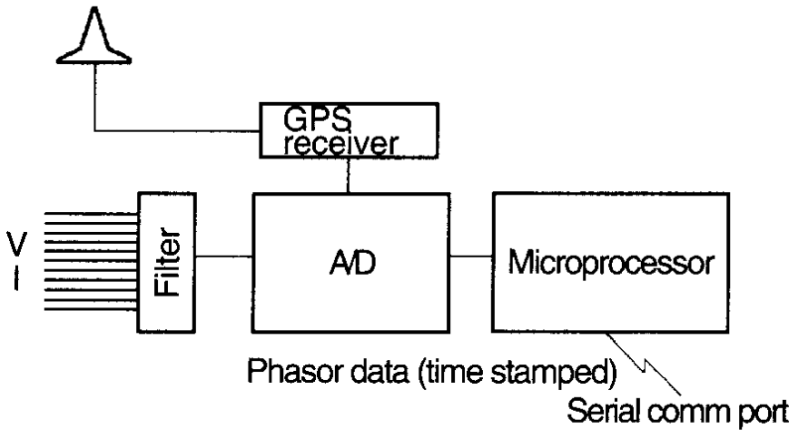

Fig. 2. – Schéma de principe d'un PMU [2]

La Fig. 2 représente les différents éléments constituant les PMUs:

- Filtre avec acquisition des mesures
- Convertisseur analogique/digital
- Récepteur GPS
- Processeur

3.2 Phasor Data Concentrator

Les Phasor Data Concentrators sont des éléments clés d'un réseau de PMUs. Ils sont situés entre les appareils de mesure et les algorithmes et applications qui consomment leurs informations. Le but d'un Phasor Data Concentrator (PDC) est de récupérer les données des PMUs et d'envoyer leurs informations. [3]

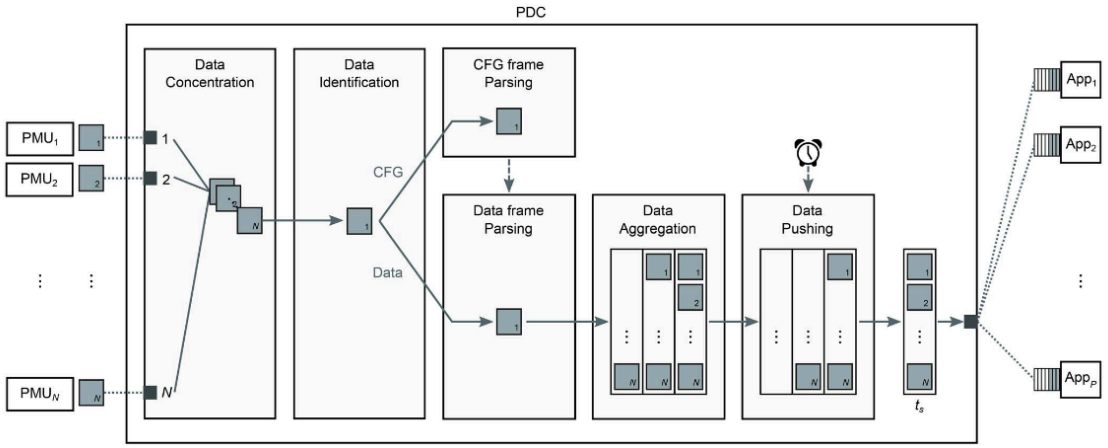

Fig. 3. – Architecture d'un PDC [3]

L'architecture d'un PDC représenté dans la Fig. 3 indique le fonctionnement d'un PDC. Celui-ci récupère les informations des multiples Phasor Measurement Units et les identifie. Les informations sont ensuite décodées à l'aide du Data Frame fourni par l'appareil puis envoyées.

La synchronisation des données des différents PMUs suit une certaine logique. Un algorithme de traitement des time-stamps t_s vient assurer cette fonction. Pour ce faire, des limites de temps sont établies au moyen de valeurs définies $t_{\min}$ et $t_{\max}$. Si $t_s < t_{\min}$, la data frame est supprimée. Si $t_{\min} \leq t_s \leq t_{\max}$, le time-stamp du buffer n'est pas mis à jour. Si $t_s > t_{\max}$, le time-stamp est remplacé par un plus récent.[3]

Ce traitement permet de minimiser le décalage des différentes valeurs récupérées par les PMUs. Un décalage trop important dans le temps fausse totalement l'interprétation des mesures et les rend obsolètes.

3.3 Protocoles et standards

La communication des données des PMUs est standardisée par la norme IEEE C37.118. [6] Cette norme définit les formats de communication pour les transmissions en temps réel. Le synchrophasor est défini comme un nombre complexe représentant une composante temporelle d'une tension ou d'un courant. Ces standards permettent d'uniformiser la communication entre les PMUs et les PDCs.

Le protocole de communication est composé de 5 frames:

- Data Frame
- Configuration Frame
- Configuration Frame
- Header Frame
- Command Frame

Les ports par défaut pour la communication en protocole Internet Protocol (IP) sont 4712 pour Transmission Control Protocol (TCP) et 4713 pour User Datagram Protocol (UDP). La communication peut également être effectuée par des ports série tels que RS232 ou RS485.

La Fig. 4 montre l'organisation d'une Data Frame avec l'emplacement des informations fournies par un PMU selon le protocole C37.118. [4]

No.	Field	Size (bytes)	Comment
1	SYNC	2	Sync byte followed by frame type and version number.
2	FRAMESIZE	2	Number of bytes in frame, defined in 6.2.
3	IDCODE	2	Stream source ID number, 16-bit integer, defined in 6.2.
4	SOC	4	SOC time stamp, defined in 6.2, for all measurements in frame.
5	FRACSEC	4	Fraction of Second and Time Quality, defined in 6.2, for all measurements in frame.
6	STAT	2	Bit-mapped flags.
7	PHASORS	4 × PHNMR or 8 × PHNMR	Phasor estimates. May be single phase or 3-phase positive, negative, or zero sequence. Four or 8 bytes each depending on the fixed 16-bit or floating-point format used, as indicated by the FORMAT field in the configuration frame. The number of values is determined by the PHNMR field in configuration 1, 2, and 3 frames.
8	FREQ	2 / 4	Frequency (fixed or floating point).
9	DFREQ	2 / 4	ROCOF (fixed or floating point).
10	ANALOG	2 × ANNMR or 4 × ANNMR	Analog data, 2 or 4 bytes per value depending on fixed or floating-point format used, as indicated by the FORMAT field in configuration 1, 2, and 3 frames. The number of values is determined by the ANNMR field in configuration 1, 2, and 3 frames.
11	DIGITAL	2 × DGNMR	Digital data, usually representing 16 digital status points (channels). The number of values is determined by the DGNMR field in configuration 1, 2, and 3 frames.
	Repeat 6–11		Fields 6–11 are repeated for as many PMUs as in NUM_PMU field in configuration frame.
12+	CHK	2	CRC-CCITT

Fig. 4. – Organisation d'une Data Frame [4]

La Fig. 5 montre le scenario de communication entre le contrôleur et le PMU. Chaque fois que les informations sont transmises depuis le PMU, ces étapes sont réalisées dans l'ordre indiqué. [4]

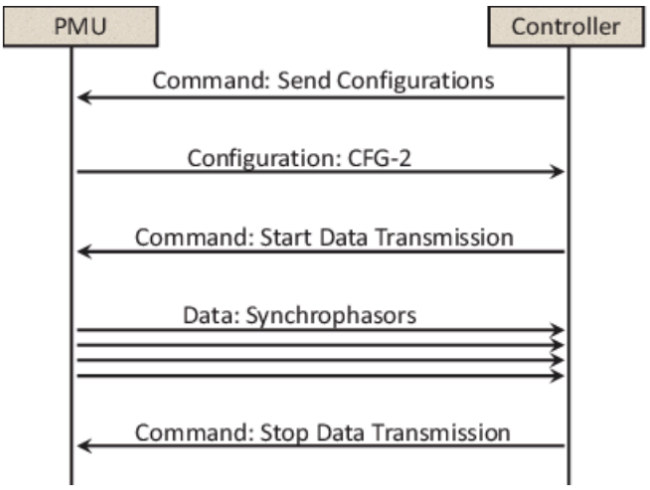

Fig. 5. – Scenario de communication [4]

3.4 Application des PMUs

Les PMUs peuvent être utilisés pour de nombreuses choses telles que l'estimation d'état, l'instabilité de tension, les marges de transfert de puissance, le calcul de paramètre de ligne, le monitoring thermique, le système de contrôle et de protection et l'analyse d'oscillation.[6]

III Câblage et configuration

Table des matières

1 Câblage du Switch et des PMUs

Les PMUs et le Switch sont interconnectés afin de permettre la remontée de mesures de tous les appareils en même temps. La structure réseau est indiquée dans la Fig. 6.

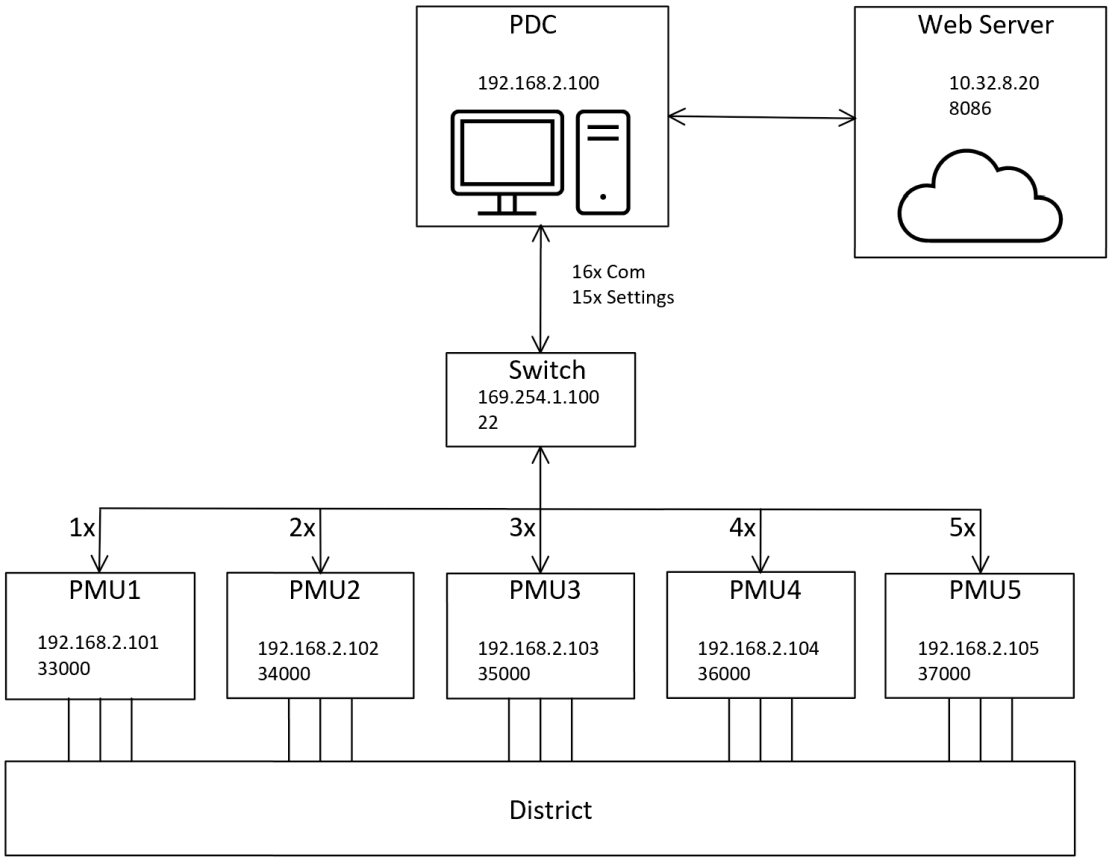

Fig. 6. – Schéma de principe du système

Les PMUs sont reliés par leur port ETH0 via un câble RJ45 à leur numéro de port correspondant sur le Switch (PMU1 - 1x, PMU2 - 2x, etc). Le port x16 du Switch permet de connecter un ordinateur pour récupérer les données envoyées. Le port x15 du Switch permet de modifier les réglages de celui-ci.

2 Câblage des PMUs

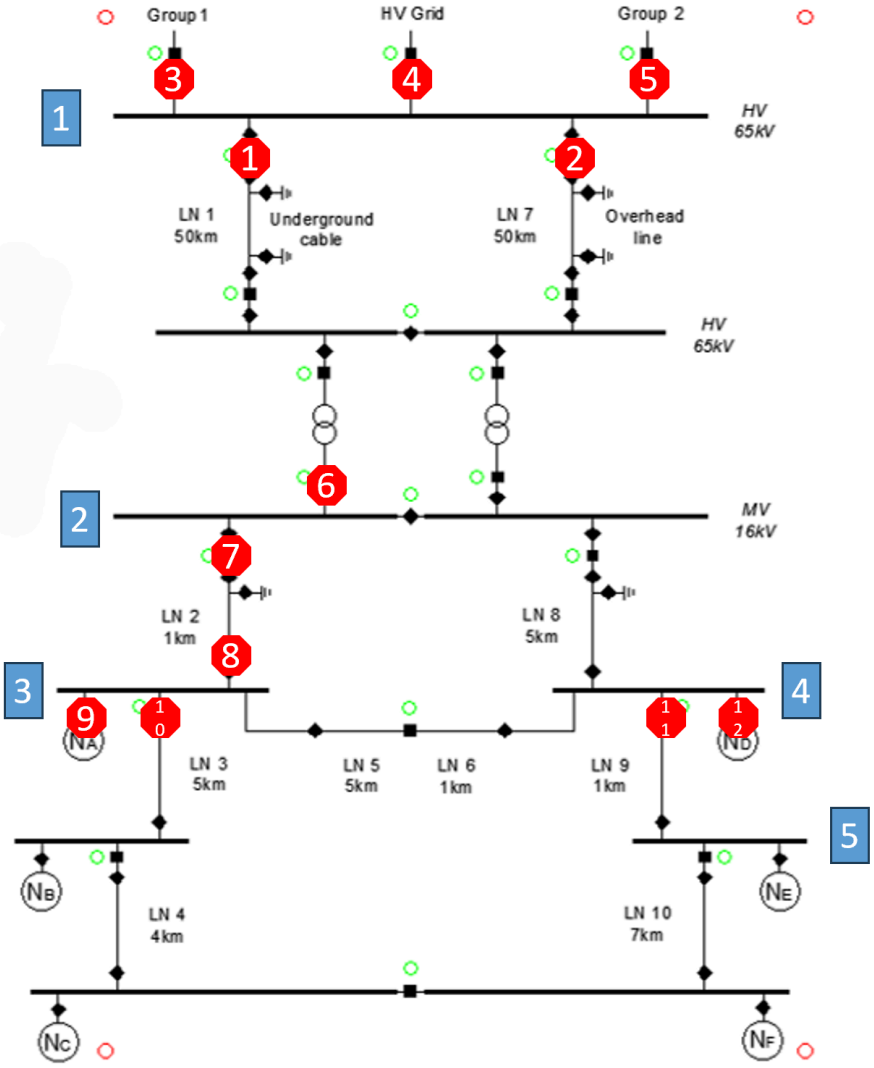

Fig. 7. – Emplacement des appareils

La Fig. 7 indique l'emplacement des appareils sur le réseau. Les rectangles bleus représentent les PMUs et leur numérotation correspond à leur identifiant (PMU0001,PMU0002,etc). Ils sont placés au niveau des bus sur lesquels ils mesurent les tensions des trois phases. Les octogones rouges représentent les sondes de courant. Celles-ci sont liées au bus et sont connectées au PMU correspondant (6 et 7 au PMU2, 11 et 12 au PMU4, etc). La numérotation des sondes de courant ainsi que la phase mesurée est également indiquée sur l'installation physique (voir Fig. 8).

Fig. 8. – Numérotation des sondes

2.1 Câblage des mesures

Avant d'effectuer tout type d'intervention, les appareils doivent être reliés à la terre à l'aide du bornier prévu à cet effet.

Chaque appareil est muni de deux cartes d'acquisition de mesures. Une carte est prévue pour les mesures de tensions et l'autre pour les mesures de courant.

2.1.1 Acquisition des tensions

Les cartes permettant de mesurer les tensions sont dotées de 4 points de connexion: neutre, phase 1, phase 2 et phase 3. Sur la carte, le neutre correspond à N, la phase 1 à 0, la phase 2 à 1 et la phase 3 à 2. Tous les appareils doivent être connectés suivant la même logique avec les phases correspondantes pour chacun.

Les détails des connexions pour les mesures de tension sont indiqués dans l'Annexe B.

2.1.2 Acquisition des courants

L'acquisition des valeurs de courant se fait au moyen de sondes. Les sondes de courant viennent se brancher en respectant la polarité dans le bornier de la carte. Pour chaque carte d'acquisition des courants, un raccordement à la terre est effectué via le connecteur « Com » (voir Chapitre VII | 1). Le code couleur pour la polarité est le suivant: bleu - positif et brun - négatif. Un connecteur multiple est placé entre les sondes et les PMUs afin de réduire le nombre de câbles apparents.

Les connexions « sondes-câble-PMU » suivent un code couleur indiqué dans l'Annexe C.

Les sondes de courant viennent se placer autour du câble à mesurer en respectant le sens de la mesure indiqué sur leur boîtier. Le sens des courants peut être modifié par la suite dans les paramètres de réglage (voir Chapitre III | 4). L'ordre d'acquisition est défini au branchement: 1 - IA1, 2 - IB1, 3 - IC1, 4 - IA2, etc. (voir Fig. 9)

Fig. 9. – Carte d'acquisition de courant

3 Configuration du Switch

Pour permettre l'accès à l'ensemble des PMUs, la communication passe par un routeur Cisco IE2000U. Les ports du routeur doivent être paramétrés pour établir la connexion à tous les appareils. Les PMUs sont considérés comme des serveurs et possèdent donc tous un port dédié. Ils communiquent avec un ordinateur qui fait office de client qui vient effectuer les requêtes aux différents. Le client possède également un port dédié.

3.1 Accès aux paramètres du switch

L'accès au contrôle du routeur peut s'effectuer de différentes manières: USB, SSH et via un dashboard web. Le paramétrage par USB et SSH s'effectue avec le logiciel Putty. L'accès par Universal Serial Bus (USB) requiert l'installation d'un driver USB spécifique. La page web est le moyen le plus simple d'accéder aux réglages.

Afin d'avoir accès à la page de configuration web, il est nécessaire de se trouver sur le même sous réseau que le Switch. Le Switch et l'ordinateur doivent être connectés physiquement à l'aide d'un câble RJ45 au port x15.

Les paramètres réseaux choisis pour ce faire sont les suivants:

IPv4 address: 169.254.1.101
Subnet Mask: 255.255.255.0
Default Gateway: 169.254.1.1

Il est ensuite possible d'effectuer une vérification de la connexion en effectuant un ping sur l'adresse du routeur (169.254.1.100) pour s'assurer que les paramètres réseau sont corrects. L'accès à l'interface web s'effectue ensuite en rentrant l'adresse IP dans une barre de recherche web. La dernière étape consiste à inscrire le nom d'utilisateur et le mot de passe respectivement **<admin>** et **<password>** .

Ces étapes donnent accès au dashboard de l'appareil qui contient une représentation de l'état des ports du switch et différents onglets de monitoring et de configuration.

3.2 Configuration des ports

Dans le menu déroulant Configure –> Port Settings, l'ensemble des ports du switch sont paramétrables. Les ports Fa0/1 à Fa/05 sont définis pour les 5 PMUs en VLAN 3. Le port Fa0/16 est défini en VLAN 3 pour la connexion à l'ordinateur qui sert également de PDC. La connexion au serveur pour les réglagles du switch s'effectue au port Fa0/15. Tous les paramètres sont modifiables à partir de cette interface. Il est possible d'ouvrir des ports supplémentaires dans le cas ou d'autres appareils venaient à être rajoutés.

3.3 Configuration du PTP

Afin de finaliser la configuration des clocks des PMUs, les commandes suivantes doivent être réalisées en Secure Shell (SSH) sur le switch.

Établir la connexion avec le switch et rentrer la commande suivante:

```
ptp mode forward
```

Enregistrer la modification:

```
copy running-config startup-config
```

Ces commandes permettent le transfert du clock du PMU master aux autres PMUs.

La procédure est réalisée correctement si les leds du switch correspondant aux ports utilisés par les PMUs clignotent en vert en même temps. La vérification peut également s'effectuer au moyen de Wireshark où les appareils envoient les paquets avec le même timestamp.

4 Configuration des PMUs

Les PMUs au nombre de 5 sont paramétrables via SSH à l'aide du logiciel Putty ou par une interface web. Deux ports Ethernet sont à disposition pour accéder au données et aux différents paramètres. Une adresse IP commune (10.11.12.13) à tous les appareils est paramétrée pour la communication sur le port ETH1. Sur le port ETH0, chaque PMU possède une adresse IP différente. L'accès à la page web de configuration peut s'effectuer depuis les deux adresses sur le port 8005 avec le nom d'utilisateur **<admin>** et le mot de passe <>.

L'attribution des adresses est la suivante:

PMU ID	ETH0	ETH1
PMU 1	192.168.2.101	10.11.12.13
PMU 2	192.168.2.102	10.11.12.13
PMU 3	192.168.2.103	10.11.12.13
PMU 4	192.168.2.103	10.11.12.13
PMU 5	192.168.2.103	10.11.12.13

Tableau 1. – Attribution des addresses des PMUs

Afin d'établir correctement la communication entre les appareils et l'ordinateur, celui-ci doit se trouver sur le même sous réseau. Pour les adresses communes: 10.11.12.13.14 avec le masque de sous réseau 255.255.255.0. Pour les autres adresses: 192.168.2.100 avec le masque de sous réseau 255.255.255.0. Les paramètres peuvent être vérifiés en effectuant un « ping » depuis le terminal de commande de l'ordinateur.

La page web permet de déterminer plusieurs paramètres de communication.

4.1 Protocole de transmission

TCP

Le protocole TCP est utilisé pour la communication. Les frames de Configuration et Header sont envoyées sur demande.

UDP Only

Le protocole UDP est utilisé pour la communication. Les frames de Configuration et Header sont envoyées sur demande.

UDP Spontaneous

Le protocole UDP est utilisé pour la communication. Les frames de Configuration et Header sont envoyées de manière spontanée chaque minute.

UDP - T

Le protocole TCP est utilisé pour recevoir les frames de commande et pour envoyer les frames de configuration et header. Le protocole UDP est utilisé pour envoyer les frames de data. Les frames de Configuration et Header sont envoyées sur demande.

UDP - U

Le protocole TCP est utilisé pour recevoir les frames de commande. Le protocole UDP est utilisé pour envoyer les frames de data, de configuration et header. Les frames de Configuration et Header sont envoyées sur demande.

Le protocole utilisé pour récupérer les données des appareils est l'UDP Spontaneous.

4.2 Débit des données

Le débit des données peut également être paramétré dans l'onglet Base Configuration Settings sous Reporting Rate. Ce paramètre détermine la fréquence à laquelle les don-

nées sont envoyées par les PMUs. Il est possible de choisir plusieurs débits: 1, 10, 25, 50 ou 100 paquets de données par seconde. Le choix du débit permet de réguler la fenêtre de précision de l'analyse du réseau. Un débit plus élevé implique également une plus grosse quantité de données à traiter et stocker.

Le choix du débit est libre suivant la précision requise.

4.3 Paramétrage des ports

Pour chaque PMU, les ports de transmission UDP doivent être paramétrés. Afin de pouvoir communiquer avec tous les PMUs, leurs ports UDP doivent être différents. L'adresse de l'hôte UDP est le même pour tous et pointe vers le client (192.168.2.100).

Ces paramètres sont modifiables dans l'onglet Communication Settings sous UDP Port et UDP Remote Host.

L'attribution des ports est la suivante:

PMU ID	IP	PORT
PMU 1	192.168.2.101	33000
PMU 2	192.168.2.102	34000
PMU 3	192.168.2.103	35000
PMU 4	192.168.2.104	36000
PMU 5	192.168.2.105	37000

Tableau 2. – Attribution des ports des PMUs

4.4 Paramétrage des clocks des PMUs

Pour que les appareils de mesure aient tous la même base de temps, un PMU donne le clock aux autres. Le PMU1 est défini comme master et les autres PMUs se calent sur son horloge. Dans Timing Settings, choisir GPS sous Time Reference. Sous Precision Time Protocol (PTP) Settings (ETH0), choisir le PTP Profile IEEE 1588 [7] et changer la Priority 1 à la valeur 5.

Suite à ces réglages, les autres PMUs (2 à 5) se synchronisent sur le clock du PMU 1. Les appareils envoient leurs données de mesures au même temps. Le timestamp des appareils est envoyé dans la dataframe avec les mesures correspondantes (voir Tableau 3). D'autres paramètres doivent être modifiés sur le switch afin de permettre la synchronisation (voir Chapitre III | 3.3).

4.5 Réception des données

Les données envoyées par les PMUs peuvent être observées à l'aide du logiciel Wireshark. Les paquets sont identifiables par leur adresse IP (voir Tableau 2) et contiennent les données mesurées selon le protocole C37.118.

```
0000  78 2d 7e 11 e6 7d 00 80  2f 35 64 53 08 00 45 00   x-~··}·· /5dS··E·
0010  00 f8 8d ce 40 00 40 11  26 0a c0 a8 02 68 c0 a8   ····@·@· &····h··
0020  02 64 7d 00 8c a0 00 e4  ca 0f aa 01 00 dc 00 04   ·d}····· ··******
0030  66 61 33 71 0f 80 2c 7f  21 f6 3d b0 c7 da c0 05   fa3q··,· !·=·····
0040  97 e2 3d a6 4f bf bf b4  17 1a 3d a3 e0 36 bf 00   ··=·O··· ··=··6··
0050  45 60 3d 87 12 b9 bf ae  7b 78 42 a9 42 e0 3e a4   E`=····· {xB·B·>·
0060  03 16 3d 17 fb b1 be d8  76 6d 3b 35 d0 60 3d a1   ··=····· vm;5·`=·
0070  a1 34 41 e1 c2 fe 3e a3  db 32 3a 38 76 e6 bd 7b   ·4A···>· ·2:8v··{
0080  f9 06 3a e8 f1 86 3e 26  be b4 3b 23 d6 b1 3d f7   ··:···>& ··;#··=·
0090  0a 2b 3a d9 0d 51 3d e1  69 58 3b 06 54 e4 3d ad   ·+:··Q=· iX;·T·=·
00a0  98 10 3c 1c bc 58 3e 5e  ec 7e 3b 23 09 06 3e 89   ··<··X>^ ·~;#··>·
00b0  9a 96 3b 99 d3 aa 3e 54  66 ac 3b 8e 96 b1 3e 0c   ··;···>T f·;···>·
00c0  91 86 3a d6 a6 f3 bd ea  63 ab 3b 5c d4 32 3e 10   ··:····· c·;\·2>·
00d0  c6 44 3b 4b 7f 88 3d c2  7a 3a 3b 0b d4 62 3a 87   ·D;K··=· z:;··b:·
00e0  2c 82 3a 89 68 17 3e 4c  d4 d1 3c b1 3c c6 3d a7   ,·:·h·>L ··<·<·=·
00f0  b6 a7 3c 07 75 53 3d a3  af 9d 42 46 5c 7c c2 25   ··<·uS=· ··BF\|·%
0100  c9 25 00 00 90 92                                 ·%····
```

Fig. 10. – Données envoyées

Dans la Fig. 10 en bleu, les datas envoyées par les PMUs contenant toutes les mesures et autres informations concernant l'appareil. Ces données peuvent ensuite être décryptée à l'aide de la description de la structure (voir Fig. 4).

```
▾ Phasors (24), notation: polar, format: floating point
    Phasor #1: "VA1             ",      0,086V ∠-119,599° alt  -0,043+j -0,075V
    Phasor #2: "VB1             ",      0,081V ∠-80,613° alt   0,013+j -0,080V
    Phasor #3: "VC1             ",      0,080V ∠-28,709° alt   0,070+j -0,038V
    Phasor #4: "V1Zero          ",      0,066V ∠-78,102° alt   0,014+j -0,065V
    Phasor #5: "IA1             ",     84,631A ∠ 18,354° alt  80,325+j 26,649A
    Phasor #6: "IB1             ",      0,037A ∠-24,223° alt   0,034+j -0,015A
    Phasor #7: "IC1             ",      0,003A ∠  4,522° alt   0,003+j  0,000A
    Phasor #8: "I1Zero          ",     28,220A ∠ 18,336° alt  26,787+j  8,878A
    Phasor #9: "IA2             ",      0,001A ∠ -3,525° alt   0,001+j -0,000A
    Phasor #10: "IB2             ",      0,002A ∠  9,330° alt   0,002+j  0,000A
    Phasor #11: "IC2             ",      0,002A ∠  6,911° alt   0,002+j  0,000A
    Phasor #12: "I2Zero          ",      0,002A ∠  6,306° alt   0,002+j  0,000A
    Phasor #13: "IA3             ",      0,002A ∠  4,857° alt   0,002+j  0,000A
    Phasor #14: "IB3             ",      0,010A ∠ 12,473° alt   0,009+j  0,002A
    Phasor #15: "IC3             ",      0,002A ∠ 15,399° alt   0,002+j  0,001A
    Phasor #16: "I3Zero          ",      0,005A ∠ 11,884° alt   0,005+j  0,001A
    Phasor #17: "IA4             ",      0,004A ∠  7,865° alt   0,004+j  0,001A
    Phasor #18: "IB4             ",      0,002A ∠ -6,557° alt   0,002+j -0,000A
    Phasor #19: "IC4             ",      0,003A ∠  8,101° alt   0,003+j  0,000A
    Phasor #20: "I4Zero          ",      0,003A ∠  5,441° alt   0,003+j  0,000A
    Phasor #21: "IA5             ",      0,002A ∠  0,059° alt   0,002+j  0,000A
    Phasor #22: "IB5             ",      0,001A ∠ 11,461° alt   0,001+j  0,000A
    Phasor #23: "IC5             ",      0,022A ∠  4,692° alt   0,022+j  0,002A
    Phasor #24: "I5Zero          ",      0,008A ∠  4,579° alt   0,008+j  0,001A
  Actual frequency value: 49,5903Hz
```

Fig. 11. – Données décryptées par Wireshark

La Fig. 11 montre les données décryptées par Wireshark à l'aide la Configuration Frame à partir de la data frame.

IV Développement

Table des matières

1 Acquisition et traitement des données

L'acquisition des données des PMUs et leur traitement se fait au moyen de plusieurs codes réalisés en langage Python. Plusieurs codes différents ont été réalisés pour effectuer différentes tâches. Le code principal permettant de récupérer les données des PMUs est le code PDC (Annexe D). Ce code gère la communication avec les PMUs, traite les données et les envoie sur la base de données. Le code SSH (Annexe G) permet de communiquer avec les réglages des PMUs. Il permet de communiquer via SSH pour effectuer différentes tâches comme mettre à jour les clocks ou effectuer un reboot des appareils en cas de problème. Dans ces deux codes principaux, des méthodes spécifiques ont été spécialement créées pour répondre aux objectifs du travail. Les méthodes principales des codes sont spécifiées dans le Chapitre IV | 4.

1.1 Acquisition

Les données sont récupérées à l'aide d'un socket qui analyse le traffic sur la connexion Ethernet sur les différents ports. Les paquets de données ne contenant pas les mesures sont triés à partir de leur taille. Les données brutes sont ensuite envoyées pour le décodage. (voir Annexe D)

1.2 Décodage des données

Les données contenant les mesures sont au format hexadécimal. La première étape est de reconstruire le paquet en fonction de son architecture. Celle-ci est légèrement différente de la Fig. 4 et est structurée de la manière suivante:

No	Field	Taille Bytes
1	Sync Word	2
2	Frame Size	2
3	ID	2
4	Timestamp	4
5	Clock Quality	1
6	Frac Sec	3
7	Bytes Sync	2
8	Phasors	24 x 8
9	Fréquence	4
10	Rate of change	4
11	Digital status word	2

Tableau 3. – Structure des la data frame

À partir de cette structure, toutes les valeurs peuvent être extraites à l'aide d'un programme Python. (voir Annexe D)

1.3 Envoi sur la base de données

Les données décodées sont ensuite envoyées sur la base de données InfluxDb comme indiqué sur la Fig. 6. Les données envoyées sont identifiables par le numéro de PMU correspondant avec le timestamp des appareils. Les différents phaseurs sont stockés avec la structure suivante:

PMU_Data –> PMU_ID (PMU0001 à PMU0005) –> Type (Tension,Courant,Freq)

Les fields de tension et de courant contiennent ensuite les valeurs d'amplitude et d'angle des phaseurs correspondants (voir Tableau 4). Le field de fréquence contient la valeur de fréquence mesurée par l'appareil de mesure.

Phasor No	Field	Type
1	VA1	_amplitude, _angle
2	VB1	_amplitude, _angle
3	VC1	_amplitude, _angle
4	V1Zero	_amplitude, _angle
5	IA1	_amplitude, _angle
6	IB1	_amplitude, _angle
7	IC1	_amplitude, _angle
8	I1Zero	_amplitude, _angle
9	IA2	_amplitude, _angle
10	IB2	_amplitude, _angle
11	IC2	_amplitude, _angle
12	I2Zero	_amplitude, _angle
13	IA3	_amplitude, _angle
14	IB3	_amplitude, _angle
15	IC3	_amplitude, _angle
16	I3Zero	_amplitude, _angle
17	IA4	_amplitude, _angle
18	IB4	_amplitude, _angle
19	IC4	_amplitude, _angle
20	I4Zero	_amplitude, _angle
21	IA5	_amplitude, _angle
22	IB5	_amplitude, _angle
23	IC5	_amplitude, _angle
24	I5Zero	_amplitude, _angle

Tableau 4. – Nomenclature des points InfluxDb

Des requêtes sont effectuées par la suite à partir de la base de données pour avoir une représentation plus claire de toutes les mesures sur Grafana. Sur le dashboard Grafana, l'emplacement des mesures de tension est indiqué. Chaque PMU mesure la tension à

un bus spécifique. Le dashboard permet d'observer en temps réel les amplitudes et les angles de chaque mesure. Les fréquences aux nœuds sont également affichées.

L'accès à InfluxDb s'effectue à l'adresse suivante: **10.32.8.20:8086** avec le nom d'utilisateur <**gridlab**> et le mot de passe <**gridlab00**> . L'accès au dashboard Grafana s'effectue à l'adresse suivante: **10.32.8.20:3000** avec le nom d'utilisateur <**admin**> et le mot de passe <**admin**> .

2 Séquence de démarrage

Une séquence de démarrage à été développée à l'exécution du code. Cette séquence permet d'assurer que les connexions sont établies correctement entre les différents appareils. Les informations concernant les adresses des appareils et les ports sont affichées. Une fois que les connexions sont établies correctement, une mise à jour automatique du clock du PMU1 est effectuée si nécessaire (voir Annexe D et Annexe G).

Les informations sont affichées de la manière suivante dans le terminal/fichier (informations concernant l'affichage au Chapitre VI | 5):

```
Connection infos
ID:  PMU1  IP:  192.168.2.101  PORT:  33000
ID:  PMU2  IP:  192.168.2.102  PORT:  34000
ID:  PMU3  IP:  192.168.2.103  PORT:  35000
ID:  PMU4  IP:  192.168.2.104  PORT:  36000
ID:  PMU5  IP:  192.168.2.105  PORT:  37000

Initialization...

Establishing connection.
PMU 192.168.2.101 connected with success!
PMU 192.168.2.102 connected with success!
PMU 192.168.2.103 connected with success!
PMU 192.168.2.104 connected with success!
PMU 192.168.2.105 connected with success!

All PMUs connected with success !


Master clock set to (UTC):  2024-08-07 07:36:54.835846+00:00

Actual time:  2024-08-07 09:36:19.648166
Time delta:  35.18725600000016 seconds

Waiting for  port1  to send data.
Waiting for  port2  to send data.
Waiting for  port3  to send data.
Waiting for  port4  to send data.
Waiting for  port5  to send data.

Processing data
```

3 Mise à jour du clock

Après la première mise en route, l'horloge du PMU1 est vérifiée toutes les heures. Si l'horloge est décalée de 0.1 seconde par rapport au temps actuel, une mise à jour de l'horloge est effectuée via SSH (Annexe G). Le PMU1 lance sa procédure de reboot et n'envoie donc plus de données pendant une courte période (~5 min). À la fin du redémarrage de l'appareil, les données sont envoyées à nouveau vers la base de données.

4 Description des méthodes

4.1 def rebootAll()

Permet de relancer les PMUs indiqués. En appelant cette méthode, une connection en SSH est établie et la commande de reboot est envoyée. La méthode de la classe ClockUpdate est utilisée dans un script nommé reboot pour redémarrer tous les PMUs en cas de problème (voir Annexe G).

Le code suivant utilise cette méthode:

```
from SSH_PMUs import ClockUpdate

pmus = ["192.168.2.101","192.168.2.102","192.168.2.103","192.168.2.104",
"192.168.2.105"]

for ip in pmus:
    ClockUpdate(ip,22,"admin","",enable=True).rebootAll()
```

Pour lancer le code depuis le Raspberry Pi, effectuer les commandes suivantes:

```
cd /home/pi/Desktop
python3 reboot.py
```

4.2 def checkTime()

Cette méthode permet d'afficher l'heure du PMU1 et la compare avec l'heure actuelle. En cas d'écart trop grand, cette méthode appelle la méthode sendTime qui met à jour l'horloge (voir Chapitre IV | 4.3). Le calcul de l'écart s'effectue en prenant en compte les délais de calculs et de communication avec les appareils (voir Annexe G).

4.3 def sendTime()

Cette méthode est appelée pour envoyer la nouvelle heure à appliquer à l'horloge du PMU. Pour mettre à jour l'heure du PMU, le setpoint horaire doit être donné 37 secondes en avance par rapport au temps réel (Exemple: setpoint 12:00:37 heure réelle 12:00:00). Une fois la méthode appelée, elle envoie le setpoint horaire au bon moment (voir Annexe G).

4.4 def checkConnectivity()

Cette méthode de la classe CheckConnection permet de vérifier si la connection avec les PMUs est correctement établie. Elle est utilisée lors de la séquence de démarrage et permet d'attendre que tous les PMUs soient connectés avant de commencer à décrypter les données (voir Annexe G).

V Applications

Table des matières

Introduction
Ce chapitre est dédié aux différentes applications développées pour utiliser les données envoyées par les PMUs. Les applications lisent les données stockées sur la base de données.

1 Calculateur d'impédances

Le calculateur d'impédance permet de déterminer les impédances des lignes pourvues d'appareils de mesure. L'implémentation de la topologie du réseau est indispensable et doit être effectuée manuellement. L'algorithme réalisé permet de calculer l'impédance des lignes en lui donnant les paramètres nécessaires: PMUID1 et PMUID2 pour les tensions, PMUID pour le courant et ProbeID pour la valeur de courant à récupérer. Le nom de la mesure peut également être indiqué. Le calculateur d'impédance est codé en orienté objet. Cela signifie que la caractérisation des lignes pour les mesures d'impédance sont à effectuer une seule fois car chaque mesure est définie comme un objet de la classe Line (voir Annexe E).

L'algorithme vient ensuite effectuer les calculs pour ressortir l'impédance de la ligne désirée. La chute de tension est calculée entre les deux PMUs indiqués puis divisée par le courant mesuré. La valeur de l'inductance de la ligne peut également être calculée à l'aide la fréquence mesurée par les PMUs (voir Fig. 12).

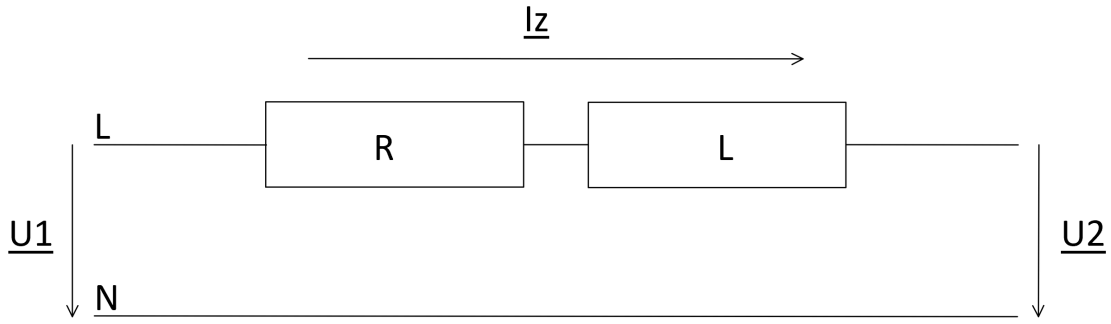

Fig. 12. – Calcul d'impédance

L'impédance est caractérisée par son inductance et par sa résistance:

$$\underline{Z} = R + j\omega L \tag{1}$$

avec

$$\omega = 2\pi f \tag{2}$$

La loi d'Ohm est utilisée dans ce cas pour déterminer l'impédance de la ligne:

$$\underline{Z} = \frac{\underline{U}}{\underline{I}} \tag{3}$$

Le calcul s'effectue en divisant la chute de tension complexe par la valeur de courant complexe passant dans la ligne. La tension U peut ainsi être remplacée par la différence de tension sur la ligne (conformément à la Fig. 12).

$$\underline{U} = \underline{U_1} - \underline{U_2} \tag{4}$$

L'impédance de la ligne peut ainsi être calculée de la manière suivante:

$$\underline{Z} = \frac{\underline{U_1} - \underline{U_2}}{\underline{I_z}} \tag{5}$$

Les valeurs de résistance et d'inductance sont calculées à partir des composantes réelles et imaginaires de l'impédance calculée. La composante réelle de l'impédance correspond à la résistance de la ligne. La composante imaginaire de l'impédance correspond à la réactance inductive de la ligne.

$$R = \Re(\underline{Z}) = |\underline{Z}| * \cos(\arg(\underline{Z})) \tag{6}$$

et

$$X_L = \Im(\underline{Z}) = |\underline{Z}| * \sin(\arg(\underline{Z})) \tag{7}$$

La valeur de l'inductance peut être calculée de la manière suivante en utilisant la fréquence mesurée:

$$L = \frac{X_L}{\omega} = \frac{\Im(\underline{Z})}{2\pi f} = |\underline{Z}| * \frac{\sin(\arg(\underline{Z}))}{2\pi f} \tag{8}$$

Voici l'exemple des paramètres à donner à l'algorithme pour calculer les impédances des 3 phases de la ligne 2:

```
1. ligne2 = Line(PMUID1=2,PMUID2=3,PMUID=3,name= "LN2",probeID= 1)
2. ligne2.getImpedance()
```

Dans la ligne 1 du code ci-dessus, les paramètres de la ligne sont donnés. La mesure de la chute de tension s'effectue entre les PMUs 2 et 3 et la mesure de courant sur la sonde 1. Le nom de la ligne doit également être entré pour la différencier des autres.

La ligne 2 du code ci-dessus appelle la méthode qui réalise les calculs décrits dans les équations (1) à équation (8). Les résultats sont ensuite affichés dans la console comme montré sur la Fig. 13.

Pour la phase de développement, les différentes impédances sont calculées et envoyées à des intervalles réguliers de 15 secondes sur la base de données.

Fig. 13. – Résultat des calculs d'impédance

2 Validation de la matrice des admittances du réseau

La matrice des admittances permet de valider la relation entre les courants et les tensions du réseau. Cette relation s'exprime par l'équation

$$I = YV \tag{9}$$

ou I représente le vecteur des courants injectés dans les lignes et V le vecteur des tensions nodales. Y représente la matrice des admittances du réseau. La matrice des admittances du réseau se construit en fonction de la topologie de celui-ci.
L'utilisation de la matrice des admittances avec les valeurs mesurées de courant sur les nœuds ou les valeurs de tension permettent de valider la relation entre les éléments.

2.1 Bus admittance matrix

La matrice des admittances permet d'effectuer des transformations sur le principe du théorème de Thévenin-Norton. La « bus admittance matrix » établie la relation entre l'injection des courants dans un nœud avec les tension de nœud d'un circuit. Cette matrice contient toutes les informations de la topologie du réseau. Elle est essentielle à la résolution de l'algorithme.
La bus admittance matrix peut être laborieuse à réaliser. Il existe un moyen plus systématique de créer cette matrice en créant une matrice « branch-to-node ». Cette matrice possède les dimensions ij définies par le réseau avec i nombre de lignes et j nombre de colonnes. Le remplissage de la matrice s'effectue en fonction du rapport entre les les lignes et les bus. Si la ligne i part du bus j, la valeur inscrite est 1. Si la ligne i arrive au bus j, la valeur est −1 et 0 dans les autres cas.

La matrice primitive des admittances est également nécessaire pour créer la « bus admittance matrix ». Celle-ci est une matrice diagonale contenant les valeurs des admittances de chaque ligne. [8]

Avec tous ces éléments, la « bus admittance matrix » peut être réalisée de la manière suivante:

$$Y_{\text{bus}} = A^T Y_{\text{pr}} A \tag{10}$$

Dans l'équation (10), A est la « branch-to-node matrix » et A^T est sa transposée.

2.1.1 Algorithme de fabrication de la bus admittance matrix

Le réseau sur lequel sont installés les appareils de mesure comporte 3 interrupteurs qui permettent de changer la topologie du réseau. L'état des interrupteurs du réseau est accessible via une requête URL. Un code à donc été réalisé pour permettre la réalisation de la « branch-to-node matrix » automatiquement et ainsi calculer la « bus admittance matrix ».

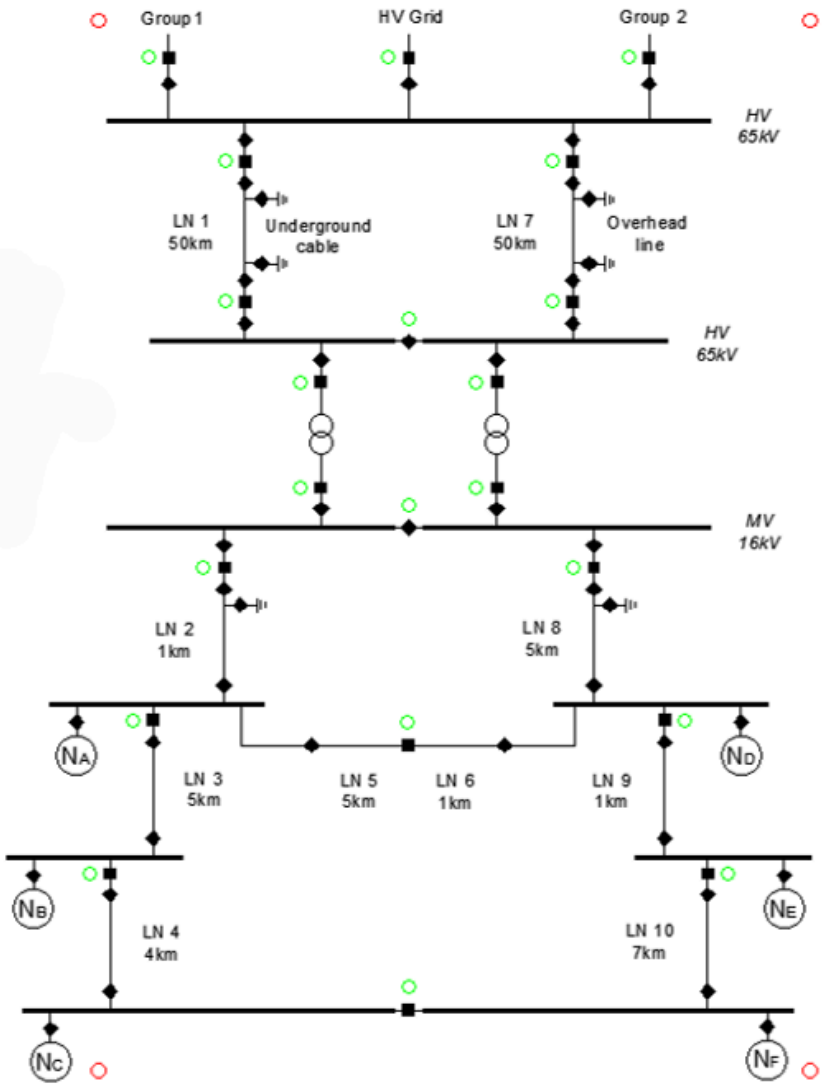

Fig. 14. – Topologie du réseau

La Fig. 14 montre la topologie du réseau où sont installés les appareils de mesure. Les carrés et losanges représentent respectivement les circuits breakers et les switchs contrôlables sur le réseau. L'état de ceux-ci permet de créer la « branch-to-node matrix ». La topologie peut être fortement impactée par la modularité de certains bus. Les bus HV 65 kV, MV 16 kV et le bus reliant les points Nc et Nf ont un impact important dans la création des matrices. Avec 3 bus séparables, cela crée 2^3 soit 8 possibilités de structures différentes pour la « branch-to-node matrix ».

Case	CB_LN4_LN10	Switch_LN1_LN7	Switch_LN2_LN8
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

Tableau 5. – Structures du réseau

Le Tableau 5 indique les différentes configurations du réseau en fonction de l'état des interrupteurs des bus mentionnés. L'état des interrupteurs est défini comme 1 fermé et 0 ouvert. Pour chaque cas, la « branch-to-node matrix » est différente. Le code Python (voir Annexe F) effectue les requêtes pour connaître l'état du réseau et crée les matrices en fonction de la structure de celui-ci.

2.2 Utilisation de la matrices des admittances

La matrice des impédances peut désormais être utilisée pour vérifier la cohérence entre les valeurs de courant mesurés et les valeurs de tension mesurées au moyen de l'équation (9). À l'inverse, la matrice des admittances peut également être vérifiée au moyen des courants de lignes et des tensions nodales mesurés.

VI Implémentation sur Raspberry

Table des matières

1 Accès au Raspberry Pi

Les différents algorithmes réalisés sont implémentés sur un Raspberry Pi 3 1GB tournant sous environnement Linux afin que l'ensemble du système puisse fonctionner de manière autonome. L'accès au Raspberry s'effectue via SSH après avoir établi une connexion au réseau câblé du Gridlab.

La communication avec le Raspberry Pi dédié au projet s'effectue via le logiciel Putty à l'adresse suivante: 10.30.4.93 port 22. Le login et mot de passe doivent être rentrés comme indiqué:

```
login as: pi
pi@10.30.4.93's password: gridlab
```

L'accès peut également se faire d'un ordinateur connecté au même réseau à l'aide du terminal de commande de la manière suivante:

```
ssh pi@10.30.4.93
```

2 Paramètres IP

Différents paramètres doivent être modifiés pour établir la connexion du Raspberry Pi avec le Switch et le réseau câblé. La connexion au réseau s'effectue directement à l'aide d'un adaptateur USB vers RJ45. La communication vers le Switch s'effectue avec le port Ethernet dédié du Raspeberry Pi vers le port x16 du Switch CISCO.

2.1 Attribution d'une adresse IP à l'adaptateur

L'adresse IP 192.168.2.100 doit être attribuée au port Ethernet du Raspberry Pi. Pour ce faire, les commandes suivantes doivent être exécutées:

```
sudo ip addr add 192.168.2.100/24 dev enxb827ebbc0b78
sudo ip link set enxb827ebbc0b78 up
```

En effectuant ces commandes, l'adresse IP est attribuée et la communication entre le Raspberry Pi et le Switch est désormais possible.

2.2 Paramétrages lors de la mise sous tension

Lors d'une remise en route suite à une interruption de l'alimentation ou d'un reboot, les paramètres modifiés se retrouvent supprimés et toutes les opérations doivent être répétées. La connexion au réseau de l'école est également interrompue et une intervention physique pour débrancher et rebrancher l'adaptateur USB/RJ45 est nécessaire. Pour résoudre ce problème, un script au démarrage de la machine a été ajouté.

Pour réaliser ces opérations, créer un shell script:

```
sudo nano /home/pi/startup_network.sh
```

Modifier le fichier de la manière suivante:

```
#! /bin/bash
sudo ip addr add 192.168.2.100/24 dev enxb827ebbc0b78
sudo ip link set enxb827ebbc0b78 up
sudo uhubctl -l 1-1 -p 2 -a off
sleep 5
sudo uhubctl -l 1-1 -p 2 -a on
sleep 10
cd /home/pi/Desktop
nohup python3 PDCGridLab.py > nohup.out 2>&1 < /dev/null &
```

Ouvrir le fichier pour paramétrer l'exécution des scripts au démarrage:

```
sudo nano/etc/systemd/system/startup_network.service
```

Puis modifier le fichier de la manière suivante:

```
[Unit]
Description=Set up network interface on startup and run PDCGridLab script
After=network.target

[Service]
ExecStart=/home/pi/startup_network.sh
ExecStartPost=/bin/sleep 10
RemainAfterExit=yes
Type=oneshot

[Install]
WantedBy=multi-user.target
```

Au démarrage, l'adresse IP du port Ethernet est paramétrée. Le port USB sur lequel est branché l'adaptateur est mis hors tension pendant 5 secondes puis la tension est rétablie. Cela simule une reconnexion physique de l'élément.

À la mise sous tension du Raspberry, celui-ci peut communiquer à nouveau avec le Switch et le réseau.

3 Accès aux PMUs via SSH

La communication aux PMUs par SSH fonctionne également par le biais du terminal du Raspberry Pi. L'accès se fait via les commandes suivantes en indiquant les adresses IP correspondantes (voir Tableau 2).

```
ssh admin@ADRESSEIP
```

En cas de problème sur un appareil de mesure, la commande suivante lance une procédure de redémarrage.

```
reboot
```

4 Modification/Mise à jour des codes

La modification ou la mise à jour des codes sur le Raspberry Pi s'effectue au moyen du logiciel FileZilla. L'accès au Raspberry s'effectue avec l'adresse <**10.30.4.93**> , le nom d'utilisateur <**pi**> , le mot de passe <**gridlab**> et le port <**22**> . Les fichiers .py sont à déposer dans **/home/pi/Desktop** . Une fois que les fichiers sont remplacés, le script doit être relancé à nouveau (voir Chapitre VI | 5).

5 Exécution du script

5.1 Exécution en premier plan

Le script principal peut être exécuté de la manière suivante en SSH:

```
cd /home/pi/Desktop
python3 PDCGridLab.py
```

En exécutant le script de cette manière, les output sont affichés dans le terminal et sont visibles par l'utilisateur. Lorsque la connexion en SSH est interrompue, le script s'arrête et une connexion doit être établie à nouveau pour permettre de relancer le script.

5.2 Exécution en arrière plan

Pour permettre au script de continuer de tourner de manière indéfinie après une interruption de la communication SSH, le script doit être lancé de la manière suivante:

```
cd /home/pi/Desktop
nohup python3 PDCGridLab.py > nohup.out 2>&1 < /dev/null &
```

Le code sera exécuté même en cas d'interruption de la connexion. Les outputs ne sont cependant plus visibles dans le terminal par l'utilisateur.

Lors de l'exécution du code en arrière plan, les outputs sont stockés dans un fichier et peuvent être consultés en effectuant la commande suivante:

```
cat nohup.out
```

Pour arrêter le script tournant en arrière plan:

```
```

```
ps aux | grep PDCGridLab.py
```

Cette commande affiche un numéro d'exécution PID.

```
kill PID
```

Cette dernière commande arrêtera le script. Pour relancer, répéter les opérations dans le terminal. Les outputs stockés dans le fichier nohup.out lors de l'exécution précédente sont écrasés.

VII Problèmes rencontrés

Table des matières

1 Sondes de courant

Les valeurs mesurées par les PMUs doivent correspondre aux valeurs réelles mesurées sur le réseau. Le réseau comporte plusieurs bus auxquels sont connectés des PMUs. Ceux-ci mesurent les tensions des 3 phases de leurs bus attribués. Chaque appareil de mesure est également doté de plusieurs sondes de courant mesurant au maximum les courants de 5 lignes. Les PMUs sont numérotés de 1 à 5 et mesurent les bus correspondants: HV 65kV, MV 16kV, LN2/LN3, LN8/LN9 et LN9/LN10 (voir Tableau 6). Le PMU 5 ne mesure aucun courant.

PMU ID	**BUS**
PMU 1	HV 65kV
PMU 2	MV 16kV
PMU 3	LN2/LN3
PMU 4	LN8/LN9
PMU 5	LN9/LN10

Tableau 6. – Attribution des bus aux PMUs

Le PMU1 mesure les courants suivants: LN1, LN7, Group1, HV Grid et Group 2.

Le PMU2 mesure les courants suivants: Transfo1 et LN2.

Le PMU3 mesure les courants suivants: LN2, NA et LN3.

Le PMU4 mesure les courants suivants: LN9 et ND.

Pour chaque mesure de courant, 3 sondes sont posées sur les 3 phases correspondantes.

La vérification des mesures de chaque courant s'effectue en mesurant chaque courant à l'aide d'une sonde prévue à cet effet et en comparant ces valeurs à celles des appareils de mesure.

Après des premiers essais de mesure de courant avec une charge purement résistive, les valeurs de courant sont faussées.

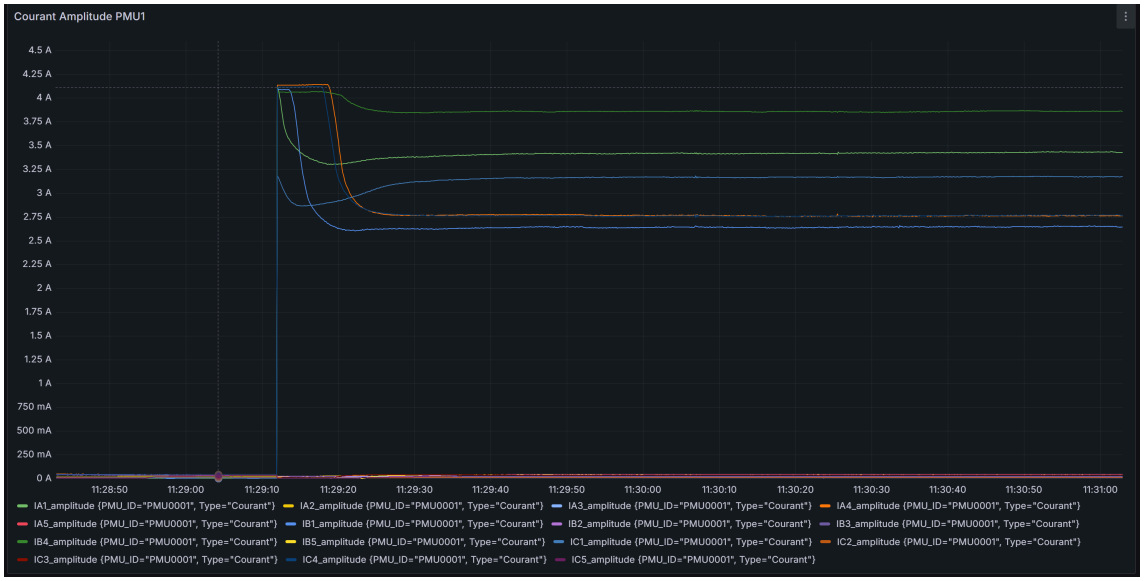

Fig. 15. – Problème sur les mesures de courant

La Fig. 15 montre le comportement des mesures de courant du PMU1. Le comportement des courants s'applique également aux autres mesures de courant. Afin de comprendre la source du problème et la résoudre, plusieurs tests ont été mis en place.

1.1 Protocoles de tests

Sur la Fig. 15, les valeurs de courant sont correctes à la mise sous tension pendant un court instant puis chutent. Les trois phases sont impactées par ce phénomène. Sur la Fig. 15, les courants de deux lignes sont mesurées et sont traversées par les mêmes courants. Les mesures des trois phases devraient être similaires une à une.

Une des solutions mentionnées relevée en effectuant de la recherche de documentation est de connecter le port COM présent sur la carte AD d'acquisition des courants. Cette solution a également été proposée par Zaphiro.

Les différentes étapes de test sont les suivantes:

- ports COMs reliés au GND commun du châssis
- ports COMs reliés au V- de leurs alimentations DC respectives
- carte d'acquisition du PMU2 déconnectée
- carte d'acquisition du PMU2 branchée et sonde de la ligne 2 débranchée

1.1.1 COMs reliés au GND du châssis

Lors de cette première phase de test, les ports COMs de tous les appareils ont été reliés au GND commun.

Fig. 16. – COMs branchés au GND commun

La Fig. 16 montre les résultats lors du branchement des ports COMs au GND commun. Les courants des trois PMUs restent corrects pendant une durée plus longue et finissent par être erronées après un certain temps.

Fig. 17. – Impact d'un reboot sur les mesures de courant

La Fig. 17 montre une fluctuation des mesures de courant lors du reboot de l'un des appareils. Lors du premier reboot du PMU2 annoté sur la Fig. 17, tous les COMs sont reliés au GND commun. Les valeurs de courant sont correctes pendant le redémarrage de l'appareil puis deviennent à nouveau erronées lorsque le PMU a correctement redémarré.

Lors du second reboot, les ports COMs sont tous déconnectés. Les valeurs de courant restent erronées et le redémarrage du PMU n'a aucun impact sur les valeurs de courant.

Cela indique que le fait de connecter les ports COMs a bel et bien un impact sur les mesures de courant.

1.1.2 COMs reliés au V- des alimentations DC

Le branchement des COMs des appareils au GND commun du châssis a un impact sur les mesures de courant comme constaté au Chapitre VII | 1.1.1. Dans cette étape, les COMs sont reliés au V- de l'alimentation DC respective de chaque PMU.

Fig. 18. – 1 COM relié au V- des alimentations DC

La Fig. 18 montre les mesures de courant lorsque le PMU2 est connecté au V-, le PMU3 au GND et le PMU1 en l'air. Seul le PMU2 connecté au V- affiche des valeurs de courant cohérentes à la réalité.

Les résultats indiquent que le fait de connecter les COMs aux alimentations respectives résoudrait le problème des courants.

Fig. 19. – 2 COMs reliés au V- des alimentations DC

La Fig. 19 montre les résultats obtenus lorsque les PMU2 et PMU3 sont reliés au V- de leur alimentation et que le PMU1 reste en l'air. Les résultats ont tendance à confirmer la solution du problème. Les courants du PMU1 restent incorrects étant donné que son port COM n'est pas relié.

Fig. 20. – Tous COMs reliés au V- des alimentations DC

La Fig. 20 montre le comportement des valeurs de courant lorsque tous les PMUs ont leurs COMs reliés à leurs alimentations respectives. Les résultats indiquent que le problème persiste. Les valeurs du PMU1 sont correctes au démarrage tandis que les valeurs des PMUs 2&3 sont erronées. Lors du reboot du PMU1, les valeurs de courant des deux autres PMUs reviennent à la normale.

1.1.3 Carte d'acquisition du PMU2 déconnectée

Trois sondes de courant du PMU2 sont placées au niveau d'éléments réactifs permettant de simuler la ligne 2 du réseau. Lors de ces tests, la carte d'acquisition de courants du PMU2 a été complètement déconnectée.

Fig. 21. – Courants sans PMU2

La Fig. 21 montre les tests réalisés en enlevant complètement l'acquisition des courants du PMU2. Les courants des PMUs 1 et 3 sont corrects jusqu'au moment ou la carte de courant du PMU2 est branchée à nouveau.

1.1.4 Carte d'acquisition branchée et sonde de la ligne 2 débranchée

Au Chapitre VII | 1.1.3, les mesures de courant semblaient être impactées par les mesures de courant du PMU2. Trois sondes de mesure de courant sont placées près d'éléments permettant de simuler la longueur de la ligne. Dans cette phase de test, les sondes problématiques sont déconnectées de la carte d'acquisition.

Fig. 22. – Courants sans les sondes de la ligne 2

La Fig. 22 montre les résultats des mesures de courant avec la configuration décrite précédemment. Au début des essais, la carte d'acquisition est branchée mais les sondes de mesure des courants de la ligne 2 ne sont pas connectées. Les courants du PMU3 subissent une perturbation au bout d'un certain temps. Le problème initial n'est pas résolu bien que les valeurs de courant restent cohérentes pendant une plus longue durée: 10 secondes à la Fig. 15 et 20 minutes à la Fig. 22.

Le problème concernant les courants n'est pas résolu par ces configurations.

1.2 Conclusion

Malgré toute la série de tests effectués, le problème concernant les mesures des courants n'est pas résolu. Entre la première et a dernière phase de test, les valeurs de courant restent correctes pendant une période de temps de plus en plus importante. Cela indique que les différentes configurations ont un impact sur les valeurs mesurées.

Des tests supplémentaires permettant de résoudre le problème n'ont pas pu être réalisés dans les temps impartis.

1.2.1 Hypothèse

Les sondes de courant sont rallongées par un câble afin de rendre l'acheminement des mesures vers les PMUs. Ce câble pourrait avoir une influence sur les mesures transmises aux PMUs. Lors de tests préliminaires avant la mise en place des PMUs sur le chariot, les courants mesurés étaient cohérents avec les valeurs de courant réelles mesurées.

Déconnecter les sondes du câble et tester les sondes connectées directement aux PMUs permettrait d'éclaircir cette hypothèse.

2 Mesure des impédances

L'algorithme des mesures des impédances a été impacté par les problèmes liés aux sondes de courant. Les valeurs d'impédance calculées par l'algorithme et calculées par le biais des configurations des lignes divergent. La mesure des impédances du réseau est également limitée par la localisation des appareils de mesure sur celui-ci.

2.1 Phases de tests

Les valeurs d'impédance ont été calculées à l'aide des caractéristiques données:

Ligne	**mOhm/km**	**uH/km**
MV	154	230
MV underground	6.8	54
MV overhead	3	63

Tableau 7. – Éléments des lignes

À partir des valeurs de lignes présentes dans le Tableau 7, les valeurs des impédances ont pu être calculées manuellement avec une fréquence commune de 50 Hz.

Ligne	Impédance
LN1	0.15 + 0.848i
LN7	0.34 + 0.989i
LN2	0.154 + 0.072i
LN8	0.77 + 0.361i
LN5	0.77 + 0.361i
LN6	0.154 + 0.072i
LN3	0.77 + 0.361i
LN9	0.154 + 0.072i
LN4	0.616 + 0.289i
LN10	1.078 + 0.505i

Tableau 8. – Impédances des lignes

Les valeurs mesurées par l'algorithme ont été comparées aux valeurs calculées dans le Tableau 8. L'instabilité des courants impliquant des valeurs d'impédance faussées ne permettent pas de comparaison directe.

2.2 Conclusion

Les problèmes concernant les mesures de courant doivent être résolues afin que l'algorithme puisse fonctionner correctement.

Ajouter d'autres PMUs et mesures de courant permettrait également de calculer l'intégralité des impédances du réseau. Cette étape est importante pour la réalisation d'algorithme d'estimation d'état.

3 Matrice des admittances

La validation de la construction de la matrice des admittances a été impactée par les problèmes liés aux mesures de courant. Les vérifications des relations de courant et de tension avec la matrice des admittances n'ont pas pu être réalisées.

4 Conclusion des problèmes rencontrés

Les instabilités observées sur les mesures de courant impactent grandement le développement des autres algorithmes. En effet, les impédances sont calculées à partir des courants et sont ensuite utilisées dans d'autres algorithmes.

VIII Conclusion

1 Résumé du projet

Ce projet consiste à implémenter des appareils de mesure de phaseurs sur un réseau simulé de distribution électrique afin de pouvoir effectuer des remontées d'informations. Le travail est séparé en deux parties distinctes.

La première partie comporte la réalisation d'une structure de réseau de communication entre les différents appareils utilisés, l'installation des appareils de mesure à des endroits stratégiques du réseau ainsi que leurs paramétrages.

La seconde partie du travail concerne l'analyse des données envoyées par les appareils de mesure, leur visualisation, leur traitement ainsi que leur utilisation dans différentes applications.

2 Comparaison avec les objectifs initiaux

Les objectifs initiaux étaient de réaliser un monitoring des données mesurées par les appareils ainsi que de réaliser un estimateur d'état linéaire.

La réalisation du monitoring des données de mesure de tous les appareils en parallèle a pu être effectuée sans problème majeur.

La réalisation de l'estimateur d'état linéaire a cependant été plus problématique étant donné les problèmes rencontrés en cours de projet. L'objectif a par ailleurs été modifié en « développement d'une matrice des admittances du réseau » pour vérifier les bonnes relations entre les courants, les tensions et les impédances du réseau.

3 Difficultés rencontrées

Les problèmes rencontrés dans ce projet étaient nombreux. La plus grosse difficulté encore non résolue à ce jour concerne les incohérences des mesures de courant (Chapitre VII | 1) qui ont impactées le développement des objectifs suivants. De par les erreurs de mesure, le calcul des impédances s'est trouvé impacté au même titre que l'algorithme de vérification. Les valeurs de courant affichées pour le monitoring sont également incohérentes.

Plusieurs autres petites difficultés ont également pu être résolues après avoir pris contact avec l'entreprise ayant fourni les appareils.

4 Perspectives futures

Afin de remplir les objectifs de ce projet à l'avenir, la plus grosse étape est d'identifier la source du problème menant à l'incohérence des mesures de courant. Cela résoudrait les problèmes liés au calcul des impédances du réseau.
L'ajout de sondes de courant supplémentaires viendrait augmenter la capacité à représenter le réseau.
La réalisation d'un estimateur d'état linéaire viendrait finaliser les objectifs initiaux de ce projet.
À l'avenir, un algorithme de détection d'erreur pourrait également être implémenté.

Acronymes

Acronyme	Abréviation
AD	Analogique Digital
DC	Direct Current
FTP	File Transfer Protocol
GPS	Global Positioning System
IP	Internet Protocol
PDC	Phasor Data Concentrator
PTP	Precision Time Protocol
SCADA	Supervisory Control And Data Acquisition
SSH	Secure Shell
TCP	Transmission Control Protocol
PTP	Precision Time Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
USB	Universal Serial Bus
VLAN	Virtual Local Area Network

Glossaire

Branch to Node Matrix

Une « branch to node matrix » est un tableau ou une matrice utilisé dans le domaine des réseaux, des télécommunications, ou des systèmes de gestion de réseau pour représenter les connexions ou les relations entre les branches (ou nœuds de départ) et les nœuds (ou destinations) dans un réseau. Cette matrice permet de visualiser et de gérer les chemins de communication, les points de connexion et les routes possibles dans un réseau complexe. Elle est souvent utilisée pour l'analyse des performances, le dépannage, et la planification des infrastructures de réseau afin d'assurer une connectivité efficace et de minimiser les conflits ou les goulots d'étranglement. [9]

DataFrame

En informatique et en science des données, un DataFrame est une structure de données bidimensionnelle, similaire à une table dans une base de données ou à une feuille de calcul, qui permet de stocker et de manipuler des données structurées de manière flexible. Chaque colonne d'un DataFrame peut contenir des données de différents types (numériques, chaînes de caractères, booléens, etc.), et chaque ligne représente un enregistrement ou une entrée unique. Les DataFrames sont couramment utilisés dans les bibliothèques de science des données comme pandas en Python et DataFrame en R, facilitant des opérations comme le filtrage, l'agrégation, la transformation et l'analyse des données. [9]

FileZilla

FileZilla est un client FTP (File Transfer Protocol) open-source qui permet aux utilisateurs de transférer des fichiers entre un ordinateur local et un serveur distant via divers protocoles tels que FTP, FTPS (FTP sécurisé) et SFTP (SSH File Transfer Protocol). Disponible pour les systèmes d'exploitation Windows, macOS et Linux, FileZilla offre une interface utilisateur graphique conviviale avec des fonctionnalités telles que le glisser-déposer, la gestion de fichiers en plusieurs onglets, et la reprise des transferts interrompus. Il est particulièrement apprécié pour sa facilité d'utilisation, sa performance et sa capacité à gérer des transferts de fichiers volumineux de manière efficace. [9]

Grafana

Grafana est une plateforme open-source de visualisation et d'analyse de données, principalement utilisée pour créer des tableaux de bord interactifs à partir de diverses sources de données. Elle est conçue pour aider les utilisateurs à surveiller, analyser et visualiser des métriques en temps réel à partir de bases de données de séries temporelles, telles qu'InfluxDB, Prometheus, ou Elasticsearch, ainsi que d'autres sources comme SQL et NoSQL. Grafana permet aux utilisateurs de concevoir des graphiques dynamiques, des diagrammes et des alertes personnalisées pour mieux comprendre les performances des systèmes et des applications. [9]

InfluxDB

InfluxDB est une base de données open-source conçue spécifiquement pour le stockage, la gestion et l'analyse de données temporelles. Développée par InfluxData, elle est optimisée pour des opérations de lecture et d'écriture à grande vitesse sur des séries temporelles, ce qui la rend particulièrement adaptée aux applications de surveillance, de suivi des performances, et d'analyse en temps réel. InfluxDB prend en charge des fonctionnalités telles que la compression des données, les requêtes SQL-like via le langage InfluxQL ou Flux, et l'intégration avec divers outils de visualisation et d'alerte. Elle est souvent utilisée pour collecter et analyser des métriques provenant de capteurs, d'applications, ou d'infrastructures IT. [9]

PTP

En informatique et en télécommunications, PTP (Precision Time Protocol) est un protocole de réseau utilisé pour synchroniser les horloges des systèmes informatiques au sein d'un réseau local. Décrit dans la norme IEEE 1588 [7], PTP permet d'obtenir une précision de synchronisation de l'ordre de la nanoseconde, ce qui est crucial pour des applications nécessitant une haute précision temporelle, telles que les systèmes financiers, les réseaux de télécommunications, et les systèmes industriels de contrôle. PTP fonctionne en échangeant des messages temporels entre les dispositifs maîtres (qui fournissent l'heure de référence) et les dispositifs esclaves (qui ajustent leur horloge pour correspondre à l'heure de référence). [9]

PuTTY

PuTTY est un client de connexion à distance gratuit et open-source pour les systèmes d'exploitation Windows et Unix. Il permet aux utilisateurs d'établir des connexions sécurisées à des serveurs distants via divers protocoles, tels que SSH (Secure Shell), Telnet, SCP (Secure Copy Protocol), et raw. PuTTY est largement utilisé pour administrer des systèmes à distance, transférer des fichiers de manière sécurisée et exécuter des commandes sur des serveurs distants. Il se distingue par sa simplicité d'utilisation et sa légèreté, tout en offrant une gamme de fonctionnalités telles que la gestion de sessions, la personnalisation des paramètres de connexion et le support de l'authentification par clé publique. [9]

Socket

En informatique, un socket est un point de terminaison pour la communication entre deux machines sur un réseau. Il représente une interface logicielle permettant aux applications de lire et d'écrire des données via un réseau, en utilisant des protocoles de communication tels que TCP (Transmission Control Protocol) ou UDP (User Datagram Protocol). Un socket est généralement défini par une combinaison d'une adresse IP et d'un numéro de port, ce qui permet aux processus d'envoyer et de recevoir des données spécifiques à cette adresse et à ce port. Les sockets sont essentiels pour le développement d'applications réseau, y compris les serveurs web, les clients de messagerie, et les jeux en ligne. [9]

Switch

En informatique et en réseaux, un switch (ou commutateur en français) est un dispositif réseau utilisé pour connecter plusieurs appareils au sein d'un réseau local (LAN). Contrairement à un hub, qui envoie les données à tous les appareils connectés, un switch dirige les données uniquement vers le périphérique destinataire en se basant sur les adresses MAC des équipements. Cela permet une utilisation plus efficace de la bande passante, réduit les collisions de données et améliore la performance du réseau. Les switches peuvent varier en taille et en fonctionnalité, allant des modèles de base pour les petits réseaux domestiques aux modèles plus avancés pour les grandes entreprises et les centres de données. [9]

Wireshark

Wireshark est un logiciel libre de capture et d'analyse de paquets réseau. Il permet aux utilisateurs d'inspecter les données échangées sur un réseau informatique en temps réel ou à partir de fichiers de capture précédemment enregistrés. Grâce à son interface graphique, Wireshark offre des fonctionnalités avancées telles que la décomposition des protocoles, le filtrage des paquets, et la reconstruction des sessions, ce qui le rend précieux pour le dépannage réseau, l'analyse de sécurité, et le développement de protocoles. Initialement connu sous le nom d'Ethereal, Wireshark est largement utilisé par les professionnels de l'informatique pour surveiller, analyser et résoudre les problèmes de réseau. [9]

Annexes

Table des matières

A Documentation

Informations concernant le câblage

Les informations concernant le câblage général des appareils est disponible au Chapitre III | 1.

Les informations concernant le câblage des mesures de tension et de courant sont disponible au Chapitre III | 2.1.

Informations concernant la configuration des appareils

Les informations concernant la configuration du switch est disponible au Chapitre III | 3.

Les informations concernant la configuration des pmus est disponible au Chapitre III | 4.

Informations concernant l'utilisation du software

Les informations concernant l'accès à la base de données sont disponibles au Chapitre IV | 1.3.

Les informations concernant l'utilisation du software sont disponibles au Chapitre VI | 5.

B Mesures de tension

PMU	Désignation	Connection	Page	Conducteur	Numéro de fil	Carte d'acquisition
PMU1	HV65 kV	430 K7	122	L1	1	0
				L2	2	1
				L3	3	2
				N	4	N
PMU2	MV 16kV	431 K4	124	L1	5	0
				L2	6	1
				L3	7	2
				N	8	N
PMU3	NA	420 K1	121	L1	9	0
				L2	10	1
				L3	11	2
				N	12	N
PMU4	ND	420 K5	121	L1	13	0
				L2	14	1
				L3	15	2
				N	16	N
PMU5	NE	420 K7	121A	L1	17	0
				L2	18	1
				L3	19	2
				N	20	N
Spare					21	
					22	
					23	
					24	

C Mesures de courant

Raccordement des courants aux PMUs et valeurs mesurées

Les bruns sont toujours par 4 avec violet turquoise et les deux autres couleurs les identifiant

PMU	Numéro schéma	Phase ID		positive	negative	port carte	mesure
PMU1	1	IA	1	bleu	blanc	AI0	LN1
		IB	1	violet	turquoise	AI1	
		IC	1	orange	blanc	AI2	
	2	IA	2	violet	turquoise	AI3	LN7
		IB	2	vert	blanc	AI4	
		IC	2	violet	turquoise	AI5	
	3	IA	3	brun	blanc	AI6	Group 1
		IB	3	violet	turquoise	AI7	
		IC	3	gris	blanc	AI8	
	4	IA	4	violet	turquoise	AI9	HV Grid
		IB	4	bleu	rouge	AI10	
		IC	4	violet	turquoise	AI11	
	5	IA	5	orange	rouge	AI12	Group 2
		IB	5	violet	tuquoise	AI13	
		IC	5	vert	rouge	AI14	
PMU2	6	IA	1	violet	turquoise	AI0	None
		IB	1	brun	rouge	AI1	
		IC	1	violet	turquoise	AI2	
	7	IA	2	gris	rouge	AI3	LN2
		IB	2	violet	turquoise	AI4	
		IC	2	bleu	noir	AI5	
PMU3	8	IA	1	violet	turquoise	AI0	LN2
		IB	1	orange	noir	AI1	
		IC	1	violet	turquoise	AI2	
	9	IA	2	vert	noir	AI3	NA
		IB	2	violet	turquoise	AI4	
		IC	2	brun	noir	AI5	
	10	IA	3	violet	turquoise	AI6	LN3
		IB	3	gris	noir	AI7	
		IC	3	violet	turquoise	AI8	
PMU4	11	IA	1	bleu	jaune	AI0	LN9
		IB	1	violet	turquoise	AI1	
		IC	1	orange	jaune	AI2	
	12	IA	2	violet	turquoise	AI3	ND
		IB	2	vert	jaune	AI4	
		IC	2	violet	turquoise	AI5	

D Code PDC

```
import socket
import threading
import sqlite3
import time
import struct
import datetime
import math
import influxdb_client
from influxdb_client import WritePrecision
from influxdb_client import WriteOptions
import threading

#Configuration
PORT1 = 34000
PORT2 = 35000
PORT3 = 36000
PORT4 = 37000
PORT5 = 33000
BUFFER_SIZE = 4096

ports = [("PMU1","192.168.2.102",PORT1),
("PMU2","192.168.2.103",PORT2),
("PMU3","192.168.2.104",PORT3),
("PMU4","192.168.2.105",PORT4),
("PMU5","192.168.2.106",PORT5)]

print('Initialization...')
print("")
time.sleep(1)
print("Connection infos")
for pmu,ip,port in ports:
    print("ID: ",pmu," IP: ",ip," PORT: ",str(port))
print("")
time.sleep(0.2)
print("Processing data")

#Function to listen on a port and update received data
def listen_on_port(port, data_dict, key):
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.bind(('0.0.0.0', port))
    while True:
        data, addr = s.recvfrom(BUFFER_SIZE)
        timestamp = time.time_ns()
        data2 = data.hex()
        if len(data2) == 440:
            pmu_number = "PMU" + data2[8:12]
            length = len(data2) - 24
            phasorValues = data2[32:length]
```

```
            frequencyValue = data2[length:length+8]
            freqHex = bytes.fromhex(frequencyValue)
            format = len(freqHex) // 4
            float_format = f">{format}f"
            value = struct.unpack(float_format, freqHex)
            frequencyPMU = value[0]
            hex = bytes.fromhex(phasorValues)
            num_floats = len(hex) // 4
            float_format = f">{num_floats}f"
            floats = struct.unpack(float_format, hex)
              decoded_data = [(floats[i], floats[i + 1]) for i in range(0,
len(floats), 2)]
            decoded_data.append(frequencyPMU)
            decoded_data.append(timestamp)
            decoded_data.append(pmu_number)
        data_dict[key] = decoded_data


latest_data = {'port1': None, 'port2': None, 'port3': None, 'port4': None,
'port5': None}

#threads to listen on ports
thread1 = threading.Thread(target=listen_on_port, args=(PORT1, latest_data,
'port1'))
thread2 = threading.Thread(target=listen_on_port, args=(PORT2, latest_data,
'port2'))
thread3 = threading.Thread(target=listen_on_port, args=(PORT3, latest_data,
'port3'))
thread4 = threading.Thread(target=listen_on_port, args=(PORT4, latest_data,
'port4'))
thread5 = threading.Thread(target=listen_on_port, args=(PORT5, latest_data,
'port5'))
thread1.daemon = True
thread2.daemon = True
thread3.daemon = True
thread4.daemon = True
thread5.daemon = True
thread1.start()
thread2.start()
thread3.start()
thread4.start()
thread5.start()

bucket = "pmu"
org = "hevs"
token               =               "x0yqZ8KBhWG-GoFgcYaHezkXCWRKsx-iS81b3-
cu6Wwh5zpXUaX3lmxHwEVhxdhQNHNLWtJPGGSa0gEO9o_tAQ=="
url = "http://10.32.9.3:8086"

client = influxdb_client.InfluxDBClient(url=url, token=token, org=org)
write_api = client.write_api(write_precision=WritePrecision.NS)
query_api = client.query_api()
```

```
phasors    =    ['VA1','VB1','VC1','V1Zero',    'IA1','IB1','IC1','I1Zero',
'IA2','IB2','IC2','I2Zero',                     'IA3','IB3','IC3','I3Zero',
'IA4','IB4','IC4','I4Zero', 'IA5','IB5','IC5','I5Zero']
type = ['Tension','Courant']

time.sleep(1)
while True:
    points = []
    for PMU in latest_data.values():
        for i in range(len(PMU)-3):
            angle = math.degrees(float(PMU[i][1]))
            amplitude = PMU[i][0]
            if i > 3:
                phaseur = type[1]
            else:
                phaseur = type[0]
            point = influxdb_client.Point("PMU_Data") \
                .tag("PMU_ID", PMU[-1]) \
                .tag("Type", phaseur) \
                .field(phasors[i] + "_amplitude", amplitude) \
                .field(phasors[i] + "_angle", angle) \
                .time(PMU[-2], WritePrecision.NS)
            points.append(point)
        point = influxdb_client.Point("PMU_Data") \
            .tag("PMU_ID", PMU[-1]) \
            .tag("Type", "Freq") \
            .field("HZ", PMU[-3]) \
            .time(PMU[-2], WritePrecision.NS)
        points.append(point)
    write_api.write(bucket=bucket, record=points)
```

E ImpedanceCalculator

```
import datetime
from influxdb_client import InfluxDBClient
import cmath
import math
from math import degrees,radians
from engineering_notation import EngNumber

def polar_to_cartesian(amplitude, angle_degrees):
    angle_radians = cmath.pi * angle_degrees / 180.0
    return cmath.rect(amplitude, angle_radians)

class Line:
                def    __init__(self,PMUID1,PMUID2,PMUID,probeID,name,bucket
=    "pmu",org    =    "hevs",token    =    "x0yqZ8KBhWG-GoFgcYaHezkXCWRKsx-
iS81b3-cu6Wwh5zpXUaX3lmxHwEVhxdhQNHNLWtJPGGSa0gEO9o_tAQ==      ",url      =
"http://10.32.8.20:8086"):
        self.bucket = bucket
        self.org = org
        self.token = token
        self.url = url
             self.client  =  InfluxDBClient(url=self.url,  token=self.token,
org=self.org)
        self.start = datetime.date(2024, 6, 1)
        self.data = {}
        self.PMUID1 = "PMU000"+str(PMUID1)
        self.PMUID2 = "PMU000"+str(PMUID2)
        self.PMUID ="PMU000"+str(PMUID)
        self.name = name
        self.probeID = str(probeID) #entrer le numéro de la sonde de courant

    def getVoltageMeasurement(self):
        dataDict1 = {}
        dataDict2 = {}
        self.data.update({self.PMUID1: dataDict1, self.PMUID2: dataDict2})
        query_api = self.client.query_api()
        query = f'''
        from(bucket: "{self.bucket}")
          |> range(start: {self.start}, stop: now())
          |> filter(fn: (r) => r["_measurement"] == "PMU_Data")
         |> filter(fn: (r) => r["PMU_ID"] == "{self.PMUID1}" or r["PMU_ID"]
== "{self.PMUID2}")
          |> filter(fn: (r) => r["Type"] == "Tension")
          |> filter(fn: (r) => r["_field"] == "VA1_angle" or r["_field"] ==
"VB1_angle" or r["_field"] == "VC1_angle" or r["_field"] == "VA1_amplitude"
or r["_field"] == "VB1_amplitude" or r["_field"] == "VC1_amplitude")
          |> last()
        '''
        tables = query_api.query(query, org=self.org)
```

```
        for table in tables:
            for value in table.records:

self.data[value["PMU_ID"]].update({value["_field"]:value["_value"]})

    def getCurrentMeasurement(self):
        amp = "_amplitude"
        ang = "_angle"

        amplitude1 = "IA" + self.probeID + amp
        amplitude2 = "IB" + self.probeID + amp
        amplitude3 = "IC" + self.probeID + amp
        angle1 = "IA" + self.probeID + ang
        angle2 = "IB" + self.probeID + ang
        angle3 = "IC" + self.probeID + ang

        query_api = self.client.query_api()
        query = f'''
        from(bucket: "{self.bucket}")
          |> range(start: {self.start}, stop: now())
          |> filter(fn: (r) => r["_measurement"] == "PMU_Data")
          |> filter(fn: (r) => r["PMU_ID"] == "{self.PMUID}")
          |> filter(fn: (r) => r["Type"] == "Courant")
          |> filter(fn: (r) => r["_field"] == "{amplitude1}" or r["_field"]
== "{angle1}" or r["_field"] == "{amplitude2}" or r["_field"] == "{angle2}"
or r["_field"] == "{amplitude3}" or r["_field"] == "{angle3}")
          |> last()
        '''
        tables = query_api.query(query, org=self.org)
        for table in tables:
            for value in table.records:

self.data[value["PMU_ID"]].update({value["_field"]:value["_value"]})

    def getImpedance(self):
        probeA = "IA" + self.probeID
        probeB = "IB" + self.probeID
        probeC = "IC" + self.probeID
        resultat = []
        complexValues = {}
        self.getVoltageMeasurement()
        self.getCurrentMeasurement()
        for key,value in self.data.items():
            complexValues.update({key: {}, key: {}})
            for keys,values in value.items():
                complexValues[key].update({keys[:3]:[]})
            for keys, values in value.items():
                complexValues[key][keys[:3]].append(values)

        processedComplex = {}

        for pmu, measurements in complexValues.items():
```

```
            processedComplex[pmu] = {}
            for measurement, (amplitude, angle) in measurements.items():
                                        processedComplex[pmu][measurement]  =
polar_to_cartesian(amplitude, angle)

        absolute_differences = {}
        for key in processedComplex[self.PMUID1].keys():
            if key in processedComplex[self.PMUID2]:
                                if  abs(processedComplex[self.PMUID1][key])  >
abs(processedComplex[self.PMUID2][key]):
                            complex_diff = processedComplex[self.PMUID1][key] -
processedComplex[self.PMUID2][key]
                        print(key,processedComplex[self.PMUID1][key])
                        print(key,processedComplex[self.PMUID2][key])
                        absolute_differences[key] = complex_diff
                        print(key,complex_diff)
                    else:
                            complex_diff = processedComplex[self.PMUID2][key] -
processedComplex[self.PMUID1][key]
                        absolute_differences[key] = complex_diff
        divided_results = {}
        #Modifier pour les valeurs de courant qui sont pas forcément IA1 IB1
IC1. faire en sorte de pouvoir remplacer par IA IB IC
        for voltage_key, current_key in zip(['VA1', 'VB1', 'VC1'], [probeA,
probeB, probeC]):
                if voltage_key in absolute_differences and current_key in
processedComplex[self.PMUID]:
                                                    divided_results[voltage_key]
=    absolute_differences[voltage_key]    /    processedComplex[self.PMUID]
[current_key]
                print("delta U",absolute_differences[voltage_key])
                print("I",abs(processedComplex[self.PMUID][current_key]))
                                print("Z",absolute_differences[voltage_key]  /
processedComplex[self.PMUID][current_key])
        for keys,values in divided_results.items():
            #print(values.real)
            (amplitude,angle) = cmath.polar(values)
            #print((amplitude,angle))
            amplitude = EngNumber(amplitude)
            angle = degrees(angle)
            Z = (amplitude,angle)
            #print("tet",divided_results[keys].real)
                            return  self.name,Z,processedComplex[self.PMUID]
[current_key],processedComplex[self.PMUID1]
["VA1"],absolute_differences["VA1"]

            #print("Z"+keys[1:]+": "+str(Z))
                   #print("Z"  +  keys[1:]  +  "  resistance:  "  +  str(Z[0]*
math.cos(radians(Z[1]))))
                  #print("Z"  +  keys[1:]  +  "  reactance:  "  +  str(Z[0]  *
math.sin(radians(Z[1]))))
            #print("")
```

```
ligne2 = Line(PMUID1=2,PMUID2=3,PMUID=3,name= "LN2",probeID= 1 )
#ligne2bis = Line(PMUID1=2,PMUID2=3,PMUID=2,name= "LN2bis",probeID= 2 )
ligne1 = Line(PMUID1=1,PMUID2=2,PMUID=1,name= "LN1",probeID= 4)
#ligne7 = Line(PMUID1=1,PMUID2=2,PMUID=1,name= "LN7",probeID= 2)
#ligne7bis = Line(PMUID1=1,PMUID2=2,PMUID=2,name= "LN7bis",probeID= 1)
#ligne9 = Line(PMUID1=4,PMUID2=5,PMUID=4,name= "LN9",probeID= 1)

print(ligne2.getImpedance()[1])
print(ligne1.getImpedance()[1])
#print(ligne2bis.getImpedance())
#print(ligne7.getImpedance())
#print(ligne7bis.getImpedance())
#print(ligne9.getImpedance())
```

F Code Load Flow

```
import requests
from collections import defaultdict
import numpy as np
from ImpedanceCalculator import Line
from math import radians
class ApiService:
    def __init__(self, username, password, url):
        self.auth = (username, password)
        self.url = url

    def fetch_data(self):
         response = requests.get(self.url, headers={'Accept': 'application/
json'}, auth=self.auth)
        if response.status_code == 200:
            return response.json()
        else:
            response.raise_for_status()


class DataProcessor:
    def __init__(self, json_data):
        self.json_data = json_data
        self.keysClass = json_data["objects"].keys()
        self.commands = self.extract_commands()

    def extract_commands(self):
        commands = []
        for x in self.get_keys(self.json_data):
            if x.endswith("value"):
                commands.append(x)
        return [commands[i] for i in range(len(commands) - 1) if (i + 1) %
2 == 0]

    def get_keys(self, dictionary):
        result = []
        for key, value in dictionary.items():
            if isinstance(value, dict):
                new_keys = self.get_keys(value)
                result.append(key)
                for innerkey in new_keys:
                    result.append(f'{key}/{innerkey}')
            else:
                result.append(key)
        return result

    def get_value_from_path_string(self, data, path_string):
        keys = path_string.split('/')
        for key in keys:
```

```
            if isinstance(data, dict) and key in data:
                data = data[key]
            else:
                return f"Key '{key}' not found or not a dictionary: {data}"
        return data

    def get_values_for_paths(self, paths):
         return [self.get_value_from_path_string(self.json_data, path) for
path in paths]

    def combine_data(self):
        results = self.get_values_for_paths(self.commands)
        combined_data = defaultdict(list)
        for command, result in zip(self.commands, results):
            first, second, third, fourth, fith, sixt = self.get_indices(command)
            key = command[first + 1:second]
          combined_data[key].append({command[third + 1:fourth]: {command[fith
+ 1:sixt]: result}})
        return combined_data

    def get_indices(self, command):
        first = command.find("/")
        second = command.find("/", first + 1)
        third = command.find("/", second + 1)
        fourth = command.find("/", third + 1)
        fith = command.find("/", fourth + 1)
        sixt = command.find("/", fith + 1)
        return first, second, third, fourth, fith, sixt


class Bus:
    def __init__(self, bus_name):
        self.bus_name = bus_name
        self.groups = defaultdict(list)

    def add_data(self, group, switch, value):
        self.groups[group].append((switch, value))

    def display_data(self):
        for group, switches in self.groups.items():
            for switch, value in switches:
                print(f"{self.bus_name} {group} {switch} {value}")

    def get_group_data(self, group):
        return self.groups.get(group, [])


class BusManager:
    def __init__(self, data_processor):
        self.data_processor = data_processor
        self.busses = self.create_busses()

    def create_busses(self):
```

```
        busses = []
        combined_data = self.data_processor.combine_data()
        for key in self.data_processor.keysClass:
            bus = Bus(key)
            for group_data in combined_data[key]:
                for group, switch_data in group_data.items():
                    for switch, value in switch_data.items():
                        bus.add_data(group, switch, value)
            busses.append(bus)
        return busses

    def display_busses(self):
        for bus in self.busses:
            bus.display_data()
            print("")

    def get_specific_data(self, bus_name, group):
        for bus in self.busses:
            if bus.bus_name == bus_name:
                return bus.get_group_data(group)
        return []


username = 'yann.zurbrugg'
password = '^SghG$bn#MpyVe2'
url   =   'https://iottest.hevs.ch/api/v1/data/bf797462-7754-4f29-9ccf-760a
15992a24/API/'

api_service = ApiService(username, password, url)
json_data = api_service.fetch_data()

data_processor = DataProcessor(json_data)
bus_manager = BusManager(data_processor)


temp_dict = {}

exceptions     =     {'CB_LN5_LN6',     'CB_LN4_LN10',     'Switch_LN1_LN7',
'Switch_LN2_LN8'}
keys_to_exclude = {'Switch_C'}

busName = ["MV","Trafo"]

groupNameMV = ["LN1_IN", "LN2_IN", "LN3_IN", "LN4_IN", "LN5_IN", "LN6_IN",
"LN7_IN",
    "LN8_IN", "LN9_IN", "LN10_IN", "LN1_OUT", "LN2_OUT", "LN3_OUT", "LN4_OUT",
    "LN7_OUT", "LN8_OUT", "LN9_OUT", "LN10_OUT", "CB_LN5_LN6", "CB_LN4_LN10",
    "Switch_LN1_LN7", "Switch_LN2_LN8"]

groupNameTrafo = ["Trafo1_IN","Trafo2_IN","Trafo1_OUT","Trafo2_OUT"]

for bus in busName:
    if bus == "MV":
```

```
        for group in groupNameMV:
            specific_data = bus_manager.get_specific_data(bus, group)
            for switch, value in specific_data:
                line_id_prefix = group.split('_')[0]

                if any(key in switch for key in keys_to_exclude):
                    continue

                if group in exceptions:
                    line_id = group
                else:
                    line_id = line_id_prefix

                if line_id in temp_dict:
                    temp_dict[line_id].append(value)
                else:
                    temp_dict[line_id] = [value]
    else:
        for group in groupNameTrafo:
            specific_data = bus_manager.get_specific_data(bus, group)
            for switch, value in specific_data:
                line_id_prefix = group.split('_')[0]

                if any(key in switch for key in keys_to_exclude):
                    continue

                if group in exceptions:
                    line_id = group
                else:
                    line_id = line_id_prefix

                if line_id in temp_dict:
                    temp_dict[line_id].append(value)
                else:
                    temp_dict[line_id] = [value]
# Convert the accumulated dictionary to a list of dictionaries
list_of_dicts = [{line_id: states} for line_id, states in temp_dict.items()]

#print(list_of_dicts)

busStatus = []
matrixTopo = ["CB_LN4_LN10","Switch_LN1_LN7","Switch_LN2_LN8"]
for group in matrixTopo:
    specific_data = bus_manager.get_specific_data("MV", group)
    busStatus.append(specific_data[0][1])

if all(busStatus) == True:
    matrixFormat = "close"
elif busStatus == [0, 0, 1]:
    matrixFormat = "2"
elif busStatus == [0, 1, 0]:
    matrixFormat = "3"
elif busStatus == [0, 1, 1]:
```

```
    matrixFormat = "4"
elif busStatus == [1, 0, 0]:
    matrixFormat = "5"
elif busStatus == [1, 0, 1]:
    matrixFormat = "6"
elif busStatus == [1, 1, 0]:
    matrixFormat = "7"
elif busStatus == [0, 0, 0]:
    matrixFormat = "open"
#uncomment
if matrixFormat == "open":
    # Initialize a 12x11 matrix filled with zeros
    matrix = np.zeros((12, 12), dtype=int)

    # Define the order of line IDs
    line_ids = ['LN1', 'LN7', 'Trafo1', 'Trafo2', 'LN2', 'LN8', 'LN5', 'LN6',
'LN3', 'LN9', 'LN4', 'LN10']

    # Fill the matrix based on the states in list_of_dicts
    for cell in list_of_dicts:
        state = all(list(cell.values())[0])
        id = list(cell.keys())[0]
        for idx, line_id in enumerate(line_ids):
            if line_id == id:
                if line_id == "LN1":
                    matrix[idx,0] = 1 if state else 0
                    matrix[idx,1] = -1 if state else 0
                elif line_id == "LN7":
                    matrix[idx, 0] = 1 if state else 0
                    matrix[idx, 2] = -1 if state else 0
                elif line_id == "Trafo1":
                    matrix[idx, 1] = 1 if state else 0
                    matrix[idx, 3] = -1 if state else 0
                elif line_id == "Trafo2":
                    matrix[idx, 2] = 1 if state else 0
                    matrix[idx, 4] = -1 if state else 0
                elif line_id == "LN2":
                    matrix[idx, 3] = 1 if state else 0
                    matrix[idx, 5] = -1 if state else 0
                elif line_id == "LN8":
                    matrix[idx, 4] = 1 if state else 0
                    matrix[idx, 6] = -1 if state else 0
                #how ?
                elif line_id == "LN5":
                    matrix[idx, 5] = 1 if state else 0
                    matrix[idx, 6] = -1 if state else 0
                #how ?
                elif line_id == "LN6":
                    matrix[idx, 6] = 1 if state else 0
                    matrix[idx, 5] = -1 if state else 0
                elif line_id == "LN3":
                    matrix[idx, 5] = 1 if state else 0
                    matrix[idx, 7] = -1 if state else 0
```

```
                elif line_id == "LN9":
                    matrix[idx, 6] = 1 if state else 0
                    matrix[idx, 8] = -1 if state else 0
                elif line_id == "LN4":
                    matrix[idx, 7] = 1 if state else 0
                    matrix[idx, 9] = -1 if state else 0
                elif line_id == "LN10":
                    matrix[idx, 8] = 1 if state else 0
                    matrix[idx, 10] = -1 if state else 0
elif matrixFormat == "close":
    # Initialize a 12x11 matrix filled with zeros
    matrix = np.zeros((12, 8), dtype=int)

    # Define the order of line IDs
    line_ids = ['LN1', 'LN7', 'Trafo1', 'Trafo2', 'LN2', 'LN8', 'LN5', 'LN6',
'LN3', 'LN9', 'LN4', 'LN10']

    # Fill the matrix based on the states in list_of_dicts
    for cell in list_of_dicts:
        state = all(list(cell.values())[0])
        id = list(cell.keys())[0]
        for idx, line_id in enumerate(line_ids):
            if line_id == id:
                if line_id == "LN1":
                    matrix[idx, 0] = 1 if state else 0
                    matrix[idx, 1] = -1 if state else 0
                elif line_id == "LN7":
                    matrix[idx, 0] = 1 if state else 0
                    matrix[idx, 1] = -1 if state else 0
                elif line_id == "Trafo1":
                    matrix[idx, 1] = 1 if state else 0
                    matrix[idx, 2] = -1 if state else 0
                elif line_id == "Trafo2":
                    matrix[idx, 1] = 1 if state else 0
                    matrix[idx, 2] = -1 if state else 0
                elif line_id == "LN2":
                    matrix[idx, 2] = 1 if state else 0
                    matrix[idx, 3] = -1 if state else 0
                elif line_id == "LN8":
                    matrix[idx, 2] = 1 if state else 0
                    matrix[idx, 4] = -1 if state else 0
                # how ?
                elif line_id == "LN5":
                    matrix[idx, 3] = 1 if state else 0
                    matrix[idx, 4] = -1 if state else 0
                # how ?
                elif line_id == "LN6":
                    matrix[idx, 4] = 1 if state else 0
                    matrix[idx, 3] = -1 if state else 0
                elif line_id == "LN3":
                    matrix[idx, 3] = 1 if state else 0
                    matrix[idx, 5] = -1 if state else 0
                elif line_id == "LN9":
```

```
                    matrix[idx, 4] = 1 if state else 0
                    matrix[idx, 6] = -1 if state else 0
                elif line_id == "LN4":
                    matrix[idx, 5] = 1 if state else 0
                    matrix[idx, 7] = -1 if state else 0
                elif line_id == "LN10":
                    matrix[idx, 6] = 1 if state else 0
                    matrix[idx, 7] = -1 if state else 0
elif matrixFormat == "2":
    # Initialize a 12x11 matrix filled with zeros
    matrix = np.zeros((12, 10), dtype=int)

    # Define the order of line IDs
    line_ids = ['LN1', 'LN7', 'Trafo1', 'Trafo2', 'LN2', 'LN8', 'LN5', 'LN6',
'LN3', 'LN9', 'LN4', 'LN10']

    # Fill the matrix based on the states in list_of_dicts
    for cell in list_of_dicts:
        state = all(list(cell.values())[0])
        id = list(cell.keys())[0]
        for idx, line_id in enumerate(line_ids):
            if line_id == id:
                if line_id == "LN1":
                    matrix[idx, 0] = 1 if state else 0
                    matrix[idx, 1] = -1 if state else 0
                elif line_id == "LN7":
                    matrix[idx, 0] = 1 if state else 0
                    matrix[idx, 2] = -1 if state else 0
                elif line_id == "Trafo1":
                    matrix[idx, 1] = 1 if state else 0
                    matrix[idx, 3] = -1 if state else 0
                elif line_id == "Trafo2":
                    matrix[idx, 2] = 1 if state else 0
                    matrix[idx, 3] = -1 if state else 0
                elif line_id == "LN2":
                    matrix[idx, 3] = 1 if state else 0
                    matrix[idx, 4] = -1 if state else 0
                elif line_id == "LN8":
                    matrix[idx, 3] = 1 if state else 0
                    matrix[idx, 5] = -1 if state else 0
                # how ?
                elif line_id == "LN5":
                    matrix[idx, 4] = 1 if state else 0
                    matrix[idx, 5] = -1 if state else 0
                # how ?
                elif line_id == "LN6":
                    matrix[idx, 5] = 1 if state else 0
                    matrix[idx, 4] = -1 if state else 0
                elif line_id == "LN3":
                    matrix[idx, 4] = 1 if state else 0
                    matrix[idx, 6] = -1 if state else 0
                elif line_id == "LN9":
                    matrix[idx, 5] = 1 if state else 0
```

```
                    matrix[idx, 7] = -1 if state else 0
                elif line_id == "LN4":
                    matrix[idx, 6] = 1 if state else 0
                    matrix[idx, 8] = -1 if state else 0
                elif line_id == "LN10":
                    matrix[idx, 7] = 1 if state else 0
                    matrix[idx, 9] = -1 if state else 0
elif matrixFormat == "3":
    # Initialize a 12x11 matrix filled with zeros
    matrix = np.zeros((12, 10), dtype=int)

    # Define the order of line IDs
    line_ids = ['LN1', 'LN7', 'Trafo1', 'Trafo2', 'LN2', 'LN8', 'LN5', 'LN6',
'LN3', 'LN9', 'LN4', 'LN10']

    # Fill the matrix based on the states in list_of_dicts
    for cell in list_of_dicts:
        state = all(list(cell.values())[0])
        id = list(cell.keys())[0]
        for idx, line_id in enumerate(line_ids):
            if line_id == id:
                if line_id == "LN1":
                    matrix[idx, 0] = 1 if state else 0
                    matrix[idx, 1] = -1 if state else 0
                elif line_id == "LN7":
                    matrix[idx, 0] = 1 if state else 0
                    matrix[idx, 1] = -1 if state else 0
                elif line_id == "Trafo1":
                    matrix[idx, 1] = 1 if state else 0
                    matrix[idx, 2] = -1 if state else 0
                elif line_id == "Trafo2":
                    matrix[idx, 1] = 1 if state else 0
                    matrix[idx, 3] = -1 if state else 0
                elif line_id == "LN2":
                    matrix[idx, 2] = 1 if state else 0
                    matrix[idx, 4] = -1 if state else 0
                elif line_id == "LN8":
                    matrix[idx, 3] = 1 if state else 0
                    matrix[idx, 5] = -1 if state else 0
                # how ?
                elif line_id == "LN5":
                    matrix[idx, 4] = 1 if state else 0
                    matrix[idx, 5] = -1 if state else 0
                # how ?
                elif line_id == "LN6":
                    matrix[idx, 5] = 1 if state else 0
                    matrix[idx, 4] = -1 if state else 0
                elif line_id == "LN3":
                    matrix[idx, 4] = 1 if state else 0
                    matrix[idx, 6] = -1 if state else 0
                elif line_id == "LN9":
                    matrix[idx, 5] = 1 if state else 0
                    matrix[idx, 7] = -1 if state else 0
```

```
                elif line_id == "LN4":
                    matrix[idx, 6] = 1 if state else 0
                    matrix[idx, 8] = -1 if state else 0
                elif line_id == "LN10":
                    matrix[idx, 7] = 1 if state else 0
                    matrix[idx, 9] = -1 if state else 0
elif matrixFormat == "4":
    # Initialize a 12x11 matrix filled with zeros
    matrix = np.zeros((12, 9), dtype=int)

    # Define the order of line IDs
    line_ids = ['LN1', 'LN7', 'Trafo1', 'Trafo2', 'LN2', 'LN8', 'LN5', 'LN6',
'LN3', 'LN9', 'LN4', 'LN10']

    # Fill the matrix based on the states in list_of_dicts
    for cell in list_of_dicts:
        state = all(list(cell.values())[0])
        id = list(cell.keys())[0]
        for idx, line_id in enumerate(line_ids):
            if line_id == id:
                if line_id == "LN1":
                    matrix[idx, 0] = 1 if state else 0
                    matrix[idx, 1] = -1 if state else 0
                elif line_id == "LN7":
                    matrix[idx, 0] = 1 if state else 0
                    matrix[idx, 1] = -1 if state else 0
                elif line_id == "Trafo1":
                    matrix[idx, 1] = 1 if state else 0
                    matrix[idx, 2] = -1 if state else 0
                elif line_id == "Trafo2":
                    matrix[idx, 1] = 1 if state else 0
                    matrix[idx, 2] = -1 if state else 0
                elif line_id == "LN2":
                    matrix[idx, 2] = 1 if state else 0
                    matrix[idx, 3] = -1 if state else 0
                elif line_id == "LN8":
                    matrix[idx, 2] = 1 if state else 0
                    matrix[idx, 4] = -1 if state else 0
                # how ?
                elif line_id == "LN5":
                    matrix[idx, 3] = 1 if state else 0
                    matrix[idx, 4] = -1 if state else 0
                # how ?
                elif line_id == "LN6":
                    matrix[idx, 4] = 1 if state else 0
                    matrix[idx, 3] = -1 if state else 0
                elif line_id == "LN3":
                    matrix[idx, 3] = 1 if state else 0
                    matrix[idx, 5] = -1 if state else 0
                elif line_id == "LN9":
                    matrix[idx, 4] = 1 if state else 0
                    matrix[idx, 6] = -1 if state else 0
                elif line_id == "LN4":
```

```
                    matrix[idx, 5] = 1 if state else 0
                    matrix[idx, 7] = -1 if state else 0
                elif line_id == "LN10":
                    matrix[idx, 6] = 1 if state else 0
                    matrix[idx, 8] = -1 if state else 0
elif matrixFormat == "5":
    # Initialize a 12x11 matrix filled with zeros
    matrix = np.zeros((12, 10), dtype=int)

    # Define the order of line IDs
    line_ids = ['LN1', 'LN7', 'Trafo1', 'Trafo2', 'LN2', 'LN8', 'LN5', 'LN6',
'LN3', 'LN9', 'LN4', 'LN10']

    # Fill the matrix based on the states in list_of_dicts
    for cell in list_of_dicts:
        state = all(list(cell.values())[0])
        id = list(cell.keys())[0]
        for idx, line_id in enumerate(line_ids):
            if line_id == id:
                if line_id == "LN1":
                    matrix[idx, 0] = 1 if state else 0
                    matrix[idx, 1] = -1 if state else 0
                elif line_id == "LN7":
                    matrix[idx, 0] = 1 if state else 0
                    matrix[idx, 2] = -1 if state else 0
                elif line_id == "Trafo1":
                    matrix[idx, 1] = 1 if state else 0
                    matrix[idx, 3] = -1 if state else 0
                elif line_id == "Trafo2":
                    matrix[idx, 2] = 1 if state else 0
                    matrix[idx, 4] = -1 if state else 0
                elif line_id == "LN2":
                    matrix[idx, 3] = 1 if state else 0
                    matrix[idx, 5] = -1 if state else 0
                elif line_id == "LN8":
                    matrix[idx, 4] = 1 if state else 0
                    matrix[idx, 6] = -1 if state else 0
                # how ?
                elif line_id == "LN5":
                    matrix[idx, 5] = 1 if state else 0
                    matrix[idx, 6] = -1 if state else 0
                # how ?
                elif line_id == "LN6":
                    matrix[idx, 6] = 1 if state else 0
                    matrix[idx, 5] = -1 if state else 0
                elif line_id == "LN3":
                    matrix[idx, 5] = 1 if state else 0
                    matrix[idx, 7] = -1 if state else 0
                elif line_id == "LN9":
                    matrix[idx, 6] = 1 if state else 0
                    matrix[idx, 8] = -1 if state else 0
                elif line_id == "LN4":
                    matrix[idx, 7] = 1 if state else 0
```

```
                    matrix[idx, 9] = -1 if state else 0
                elif line_id == "LN10":
                    matrix[idx, 8] = 1 if state else 0
                    matrix[idx, 9] = -1 if state else 0
elif matrixFormat == "6":
    # Initialize a 12x11 matrix filled with zeros
    matrix = np.zeros((12, 9), dtype=int)

    # Define the order of line IDs
    line_ids = ['LN1', 'LN7', 'Trafo1', 'Trafo2', 'LN2', 'LN8', 'LN5', 'LN6',
'LN3', 'LN9', 'LN4', 'LN10']

    # Fill the matrix based on the states in list_of_dicts
    for cell in list_of_dicts:
        state = all(list(cell.values())[0])
        id = list(cell.keys())[0]
        for idx, line_id in enumerate(line_ids):
            if line_id == id:
                if line_id == "LN1":
                    matrix[idx, 0] = 1 if state else 0
                    matrix[idx, 1] = -1 if state else 0
                elif line_id == "LN7":
                    matrix[idx, 0] = 1 if state else 0
                    matrix[idx, 2] = -1 if state else 0
                elif line_id == "Trafo1":
                    matrix[idx, 1] = 1 if state else 0
                    matrix[idx, 3] = -1 if state else 0
                elif line_id == "Trafo2":
                    matrix[idx, 2] = 1 if state else 0
                    matrix[idx, 3] = -1 if state else 0
                elif line_id == "LN2":
                    matrix[idx, 3] = 1 if state else 0
                    matrix[idx, 4] = -1 if state else 0
                elif line_id == "LN8":
                    matrix[idx, 3] = 1 if state else 0
                    matrix[idx, 5] = -1 if state else 0
                # how ?
                elif line_id == "LN5":
                    matrix[idx, 4] = 1 if state else 0
                    matrix[idx, 5] = -1 if state else 0
                # how ?
                elif line_id == "LN6":
                    matrix[idx, 5] = 1 if state else 0
                    matrix[idx, 4] = -1 if state else 0
                elif line_id == "LN3":
                    matrix[idx, 4] = 1 if state else 0
                    matrix[idx, 6] = -1 if state else 0
                elif line_id == "LN9":
                    matrix[idx, 5] = 1 if state else 0
                    matrix[idx, 7] = -1 if state else 0
                elif line_id == "LN4":
                    matrix[idx, 6] = 1 if state else 0
                    matrix[idx, 8] = -1 if state else 0
```

```
                elif line_id == "LN10":
                    matrix[idx, 7] = 1 if state else 0
                    matrix[idx, 8] = -1 if state else 0
elif matrixFormat == "7":
    # Initialize a 12x11 matrix filled with zeros
    matrix = np.zeros((12, 9), dtype=int)

    # Define the order of line IDs
    line_ids = ['LN1', 'LN7', 'Trafo1', 'Trafo2', 'LN2', 'LN8', 'LN5', 'LN6',
'LN3', 'LN9', 'LN4', 'LN10']

    # Fill the matrix based on the states in list_of_dicts
    for cell in list_of_dicts:
        state = all(list(cell.values())[0])
        id = list(cell.keys())[0]
        for idx, line_id in enumerate(line_ids):
            if line_id == id:
                if line_id == "LN1":
                    matrix[idx, 0] = 1 if state else 0
                    matrix[idx, 1] = -1 if state else 0
                elif line_id == "LN7":
                    matrix[idx, 0] = 1 if state else 0
                    matrix[idx, 1] = -1 if state else 0
                elif line_id == "Trafo1":
                    matrix[idx, 1] = 1 if state else 0
                    matrix[idx, 2] = -1 if state else 0
                elif line_id == "Trafo2":
                    matrix[idx, 1] = 1 if state else 0
                    matrix[idx, 2] = -1 if state else 0
                elif line_id == "LN2":
                    matrix[idx, 2] = 1 if state else 0
                    matrix[idx, 3] = -1 if state else 0
                elif line_id == "LN8":
                    matrix[idx, 2] = 1 if state else 0
                    matrix[idx, 4] = -1 if state else 0
                # how ?
                elif line_id == "LN5":
                    matrix[idx, 3] = 1 if state else 0
                    matrix[idx, 4] = -1 if state else 0
                # how ?
                elif line_id == "LN6":
                    matrix[idx, 4] = 1 if state else 0
                    matrix[idx, 3] = -1 if state else 0
                elif line_id == "LN3":
                    matrix[idx, 3] = 1 if state else 0
                    matrix[idx, 5] = -1 if state else 0
                elif line_id == "LN9":
                    matrix[idx, 4] = 1 if state else 0
                    matrix[idx, 6] = -1 if state else 0
                elif line_id == "LN4":
                    matrix[idx, 5] = 1 if state else 0
                    matrix[idx, 7] = -1 if state else 0
                elif line_id == "LN10":
```

```
                    matrix[idx, 6] = 1 if state else 0
                    matrix[idx, 8] = -1 if state else 0

#uncomment
# Print the resulting matrix
#   HV65 -- HV65G -- HV65D -- MV16G -- MV16D -- LN2/LN3 -- LN8/LN9 -- LN3/
LN4 -- LN9/LN10 -- BOTTG -- BOTTD
#LN1
#LN7
#Trasfo1
#Trasfo2
#LN2
#LN8
#LN5
#LN6
#LN3
#LN9
#LN4
#LN10
print(matrixFormat,matrixTopo[0],busStatus[0],matrixTopo[1],busStatus[1]
,matrixTopo[2],busStatus[2])
#print(matrix)
print(matrix)
#calculate all the line currents
ln1 = Line(1,2,1,1,"courant LN1")
ln7 = Line(1,2,1,2,"courant LN7")
trafo1 = Line(1,2,2,1,"courant trafo1")
trafo2 = 0
ln2 = Line(1,2,2,2,"courant LN2")
ln3 = Line(2,3,3,3,"courant LN3")
ln8 = 0
ln4 = 0
ln5 = 0
ln6 = 0
ln10 = 0
na = Line(3,2,3,2,"courant NA")

ln9 = Line(1,4,4,1,"courant LN9")
zln1 = 0.15+0.989j
zac = ln1.getImpedance()[1]
zac = complex(zac[0],radians(zac[1]))
#print(zln1)
#print(zac)
currList            =            [ln1.getImpedance()[2],ln7.getImpedance()
[2],trafo1.getImpedance()[2],trafo2,ln2.getImpedance()
[2],ln8,ln5,ln6,ln3.getImpedance()[2],ln9.getImpedance()[2],ln4,ln10]
voltList  =  [ln1.getImpedance()[3],na.getImpedance()[3]+ln1.getImpedance()
[4]*(zac-zln1)/zac,0,ln3.getImpedance()[3],0,na.getImpedance()
[3],0,0,0,0,0]
#print(currList[0],currList[2],currList[4],na.getImpedance()[2])
yValues = [0.202264281206682-1.14346740308844j
,0.310865385230786-0.904252547038961j
,0.266151467236318-0.273132489327762j
```

```
,0.203355341959483+0.25408826320875j
,5.3220786345758-2.49688429446819j
,1.0643838338716-0.499403365585885j
,1.0643838338716-0.499403365585885j
,5.3220786345758-2.49688429446819j
,1.0643838338716-0.499403365585885j
,5.3220786345758-2.49688429446819j
,1.33046982561942-0.624262489770747j
,0.760275251860482-0.356715788161886j]
ypr = np.zeros((12,12),dtype = complex)
for idx,Y in enumerate(yValues):
    ypr[idx,idx] = Y

ipr = np.zeros((12, 1), dtype=complex)
v = np.zeros((12, 1), dtype=complex)
for i,cur in enumerate(currList):
    if cur != 0:
        ipr[i] = cur

for i,volt in enumerate(voltList):
    if volt != 0:
        v[i] = volt

PQ = np.zeros((24, 1))
#print(matrix.shape)
#print(ipr.shape)
I = np.dot(matrix.T, ipr)
vpr = np.dot(matrix,v)
#print(I)
#print(vpr)
#print(ypr)
ybusInt = np.dot(matrix.T,ypr)
ybus = np.dot(ybusInt,matrix)
S = np.multiply(v,I.conj())
print(ybus)
print(v)
print(I)
#print(I.conj())
print(S)
for idx,complex in enumerate(S):
    PQ[idx] = complex.real
    PQ[idx+12] = complex.imag
#print(vpr.shape)
#print(ln1.getImpedance()[4])
print(PQ)
```

G Communication SSH

```
import datetime
import paramiko
import time

class ClockUpdate:
    def __init__(self,hostname,port,username,password,enable):
        self.hostname = hostname
        self.port = port
        self.username = username
        self.password=password
        self.enable = enable

    def sendTime(self):
        print("")
        print("Updating PMU Master Clock",flush=True)
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(self.hostname,self.port,self.username,self.password)
        print("Waiting to set time...",flush=True)
        print("",flush=True)
        while True:
            now = datetime.datetime.now()
            micro = now.microsecond // 1000
            hour = str(now.hour - 2)
            minute = str(now.minute)
            sec = now.second + 37
            if len(hour) < 2:
                hour = "0" + hour
            if len(minute) < 2:
                minute = "0" + minute

            if (micro == 000) and (now.second < 23):
                   ssh.exec_command('hwclock --set --date ' + hour + ":" +
minute +":"+str(sec))
                break

        ssh.exec_command('reboot')
        time.sleep(1)
        ssh.close()
            print("Master clock set to: ",str(hour) + ":" + str(minute)
+":"+str(sec),flush=True)
        print("",flush=True)
        while True:
            time.sleep(5)
            try:
                        ssh.connect(self.hostname, self.port, self.username,
self.password)
                ssh.close()
```

```
                self.enable = True
                time.sleep(5)
                break
            except:
                print("PMU Master is rebooting...",flush=True)
        print("",flush=True)
        print("PMU Master rebooted successfully !",flush=True)
        print("",flush=True)
        print("Processing data",flush=True)
        time.sleep(3)

    def rebootAll(self):
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(self.hostname, self.port, self.username, self.password)
        time.sleep(3)
        ssh.exec_command('reboot')
        print("PMU"+ self.hostname[-1]," has rebooted.",flush=True)
        time.sleep(1)
        ssh.close()

    def checkTime(self):
        print("")
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(self.hostname, self.port, self.username, self.password)

        stdin, stdout, stderr = ssh.exec_command('hwclock --show')
        output = stdout.read().decode()

        print("Master clock set to (UTC): ",output,flush=True)
        print("Actual time: ",datetime.datetime.now(),flush=True)
       print("Time delta: ",abs((int(output[14:16]) * 60 + int(output[17:19])
+  float(output[20:26])/1000000)  -  (datetime.datetime.now().minute  *  60
+  datetime.datetime.now().second  +  datetime.datetime.now().microsecond  /
1e6)),"seconds",flush=True)

        print("",flush=True)
        t_0 = time.time_ns()

        stdin, stdout, stderr = ssh.exec_command('hwclock --show')
        output = stdout.read().decode()
        tt = abs((t_0 - time.time_ns())/1e9)
        #print(tt)
               #print(abs((int(output[14:16])  *  60  +  int(output[17:19])
+  float(output[20:26])/1000000)  -  (datetime.datetime.now().minute  *  60
+  datetime.datetime.now().second  +  datetime.datetime.now().microsecond  /
1e6)))
           #print(tt  +  abs((int(output[14:16])  *  60  +  int(output[17:19])
+  float(output[20:26])/1000000)  -  (datetime.datetime.now().minute  *  60
+  datetime.datetime.now().second  +  datetime.datetime.now().microsecond  /
1e6)))
            if  tt  +  abs((int(output[14:16])  *  60  +  int(output[17:19])
```

```
+ float(output[20:26])/1000000) - (datetime.datetime.now().minute * 60 +
datetime.datetime.now().second + datetime.datetime.now().microsecond / 1e6))
> 37.03:
                print("Automatically updating Master clock due to time
drift",flush=True)
            print("",flush=True)
            ssh.close()
            self.sendTime()
        else:
            self.enable = True
            return self.enable
              #test = abs((int(output[14:16]) * 60 + int(output[17:19])) -
(datetime.datetime.now().minute * 60 + datetime.datetime.now().second))
            #print("test", test)
        # else:
        #     choice = input("Still wish to update ? y/n: ")
        #     if choice == "y":
        #         ssh.close()
        #         self.sendTime()
        #     elif choice == "n":
        #         print("")
        #         print("Keeping actual time settings")
        #         print("")
        #         print("Processing data")
        #         print("")
        #         ssh.close()
class CheckConnection:
                                                    hostnames             =
["192.168.2.101","192.168.2.102","192.168.2.103","192.168.2.104",
    "192.168.2.105"]

    def __init__(self,hostname = hostnames,port=22,username="admin",
    password=""):
        self.hostname = hostname
        self.port = port
        self.username = username
        self.password= password

    def checkConnectivity(self):
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        while True:
            all_connected = []
            for ip in self.hostnames:
                try:
                    ssh.connect(ip, self.port, self.username,
                     self.password)
                    print("PMU " + ip + " connected with success!")
                    all_connected.append(True)
                except:
                    print("PMU " + ip + " failed to connect :(")
                    print("")
                    all_connected.append(False)
```

```
            finally:
                ssh.close()

        if all(all_connected):
            print("")
            print("All PMUs connected with success !")
            break
        else:
            print("Trying again in 5 seconds, check your connections.")
            print("")
            time.sleep(5)
            continue
```

Bibliographie

[1] « THE 17 GOALS | Sustainable Development ». Consulté le: 17 mai 2024. [En ligne]. Disponible sur: https://sdgs.un.org/goals

[2] A. Phadke, « Synchronized Phasor Measurements in Power Systems », *IEEE Computer Applications in Power*, vol. 6, nº 2, p. 10-15, avr. 1993, doi: 10.1109/67.207465.

[3] A. Derviškadić, P. Romano, M. Pignati, et M. Paolone, « Architecture and Experimental Validation of a Low-Latency Phasor Data Concentrator », *IEEE Transactions on Smart Grid*, vol. 9, nº 4, p. 2885-2893, juill. 2018, doi: 10.1109/TSG.2016.2622725.

[4] « IEEE C37.118 Protocol ». Consulté le: 16 mai 2024. [En ligne]. Disponible sur: https://www.typhoon-hil.com/documentation/typhoon-hil-software-manual/References/c37_118_protocol.html

[5] E. Dusabimana et S.-G. Yoon, « A Survey on the Micro-Phasor Measurement Unit in Distribution Networks », *Electronics*, vol. 9, nº 2, p. 305-306, févr. 2020, doi: 10.3390/electronics9020305.

[6] K. E. Martin *et al.*, « Exploring the IEEE Standard C37.118–2005 Synchrophasors for Power Systems », *IEEE Transactions on Power Delivery*, vol. 23, nº 4, p. 1805-1811, oct. 2008, doi: 10.1109/TPWRD.2007.916092.

[7] J. C. Eidson, *Measurement, Control, and Communication Using IEEE 1588*. Springer Science & Business Media, 2006.

[8] F. Sossan, « Load Flows », 2023.

[9] « Open AI, (2024) ChatGPT (version 3.5) ». [En ligne]. Disponible sur: https://chat.openai.com/chat