J3eA, Journal sur l'enseignement des sciences et technologies de l'information et des systèmes
J3eA 24 (2025) 16
https://doi.org/10.1051/j3ea/20250016
https://www.j3ea.org

Des ressources aux SAÉ : Étude et mise en œuvre du protocole Modbus*

Vincent Thomas[1*], Sophie Dupuis[1], Bertrand Gélis[1]

[1] IUT de Montpellier, Département GEII, 99 avenue D'Occitanie, 34296 Montpellier, France

*Auteur de correspondance : vincent.thomas@umontpellier.fr

Résumé : *Cet article a pour objectif de présenter l'enchaînement pédagogique entre des ressources et des Situations d'Apprentissage et d'Évaluation, en deuxième puis troisième année de Bachelor Universitaire de Technologie Génie Électrique et Informatique Industrielle, avec pour support le protocole Modbus.*

Mots clés : Informatique industrielle, Modbus, C++/Qt, réseau, SAÉ.

Abstract:
This article aims to present the pedagogical progression between instructional resources and Learning and Assessment Situations (LAS) implemented during the second and third years of the Bachelor's degree in Electrical Engineering and Industrial Computing. The Modbus protocol serves as the central technical support throughout this sequence.

Keywords: Industrial Computing, Modbus, C++/Qt, Networking

1. Introduction

Le Bachelor Universitaire de Technologie (BUT) Génie Électrique et Informatique Industrielle (GEII) comporte trois parcours :

- ESE : Électronique et Systèmes Embarqués,
- EME : Électricité et Maîtrise de l'Énergie,
- AII : Automatisme et Informatique Industrielle.

Par ailleurs, la réforme du BUT vise à introduire les Situations d'Apprentissage et d'Évaluation (SAÉ) dans le parcours didactique des étudiants, afin de les placer dans une situation professionnelle. Les étudiants sont amenés à se servir des apprentissages critiques acquis dans le cadre des ressources (nouveau nom des modules, composés de CM, TD et TP) de la formation, afin d'atteindre un objectif précis, reflétant une problématique industrielle.

Dans cette optique, nous avons créé une série de ressources puis SAÉ sur une thématique commune : le protocole Modbus. À partir des compétences acquises lors des ressources, non seulement en ce qui concerne le protocole Modbus, mais également la création d'Interface Homme/Machine (IHM) entre autres, le but final est de communiquer avec des automates Schneider par le protocole Modbus-TCP.

La suite du papier est organisée somme suit. Les ressources sont présentées dans la section 2. Les trois sujets de SAÉ sont présentés dans la section 3. Enfin, on dresse une conclusion dans la section 4.

2. Ressources

Nous avons monté trois ressources. Une première ressource de deuxième année (semestre 3) de spécialité commune aux parcours EME et AII concerne le réseau[1] et plus précisément la prise en main du protocole Modbus, l'analyse de trames principalement. Puis une ressource de troisième année (semestre 5), en tronc commun aux trois parcours, concerne l'IOT et, plus précisément, le traitement de bases de données. Enfin, une ressource de troisième année de spécialité commune aux parcours ESE et AII a pour objectif d'initier les étudiants à la programmation orientée objet.

2.1. Ressource réseau (spé EME et AII): protocole Modbus/Modbus-TCP

Après une présentation du protocole Modbus RTU et TCP, les étudiants sont amenés au cours d'un TD à analyser les trames de requêtes Modbus-TCP, afin de consolider les notions vues précédemment. Un simulateur de serveur Modbus mutualisé écrit en Python avec la librairie pyModbusTCP [1] permet de répondre aux requêtes des étudiants. Les étudiants effectuent leurs requêtes avec le logiciel mbpoll et l'analyse de trames se fait avec le logiciel Wireshark [2] (cf. Figure 1).

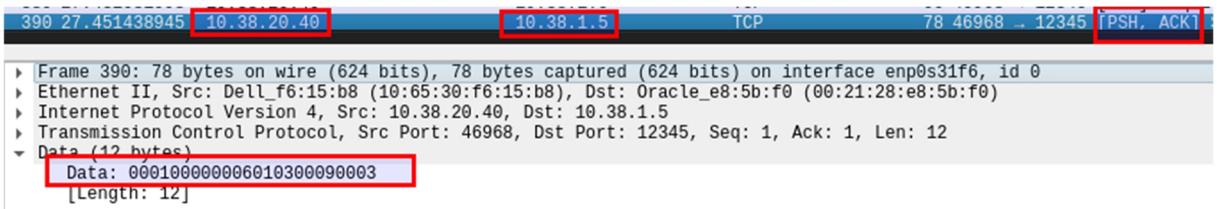


Figure 1. Trame capturée avec Wireshark

Pour mbpoll, l'extrait de de documentation suivant est donné :

[1] Cette ressource étant la seule qui a lieu en 2 année, il sera utile d'effectuer quelques rappels en début de SAÉ

usage : mbpoll [options] device|host [writevalues...] [options]

Où les options sont :

-p : numéro du port. Par défaut 502

-a : numéro/ID de l'esclave. Par défaut 1

-r : le numéro du registre. Par défaut 1

-c : le nombre de valeurs/registres à lire.

-t : une valeur correspondant à un code fonction

A titre d'exemples :

- Pour lire 3 valeurs à partir du registre 10 sur l'esclave 1 de l'hôte 10.38.1.5 sur le port 12345 :

  mbpoll 10.38.1.5 -p 12345 -a 1 -r 10 -c 2

- Pour écrire la valeur 42 dans le registre 12 sur l'esclave 1 de l'hôte 10.38.1.5 sur le port 12345 :

  mbpoll 10.38.1.5 -p 12345 -a 1 -r 12 42

L'objectif est de retrouver les éléments composant une trame en termes d'adresses d'esclave, codes fonctions, questions/réponses, tailles et numérotations des trames, types de données. Les étudiants sont amenés avec des essais itératifs, et à force de changements de paramètres, à identifier tous les champs (cf. Figure 2).

La manipulation permet aussi de faire des rappels de communication TCP/IP en termes de configuration réseau, adressage IP, ports, source/destination, acquittements, ...

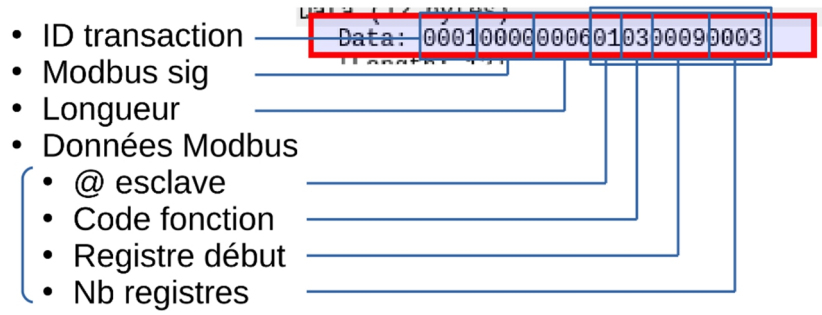


Figure 2. Détail d'une trame Modbus-TCP

En fin de séance, le simulateur est remplacé par un automate Schneider M340 pilotant une maquette dite "tri de colis" (cf. Figure 3). Le programme est donné aux étudiants et, après une analyse rapide des entrées/sorties, les étudiants s'essaient au pilotage des vérins ou de la vitesse du moteur, ainsi qu'à la lecture des détecteurs optiques ou du code barre situé sur les colis.

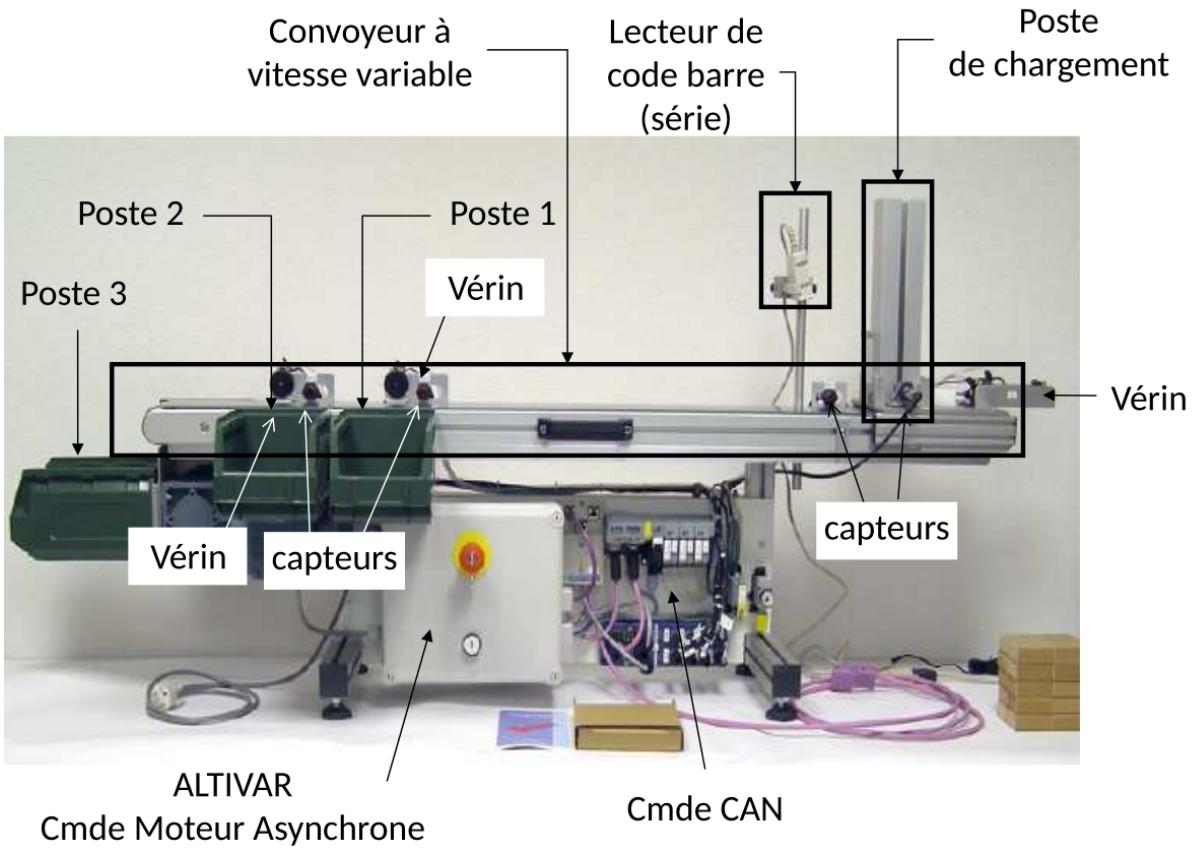


Figure 3. Photographie de la maquette "tri de colis"

Ensuite, en TP, il est demandé tout d'abord aux étudiants d'étudier la bibliothèque libmodbus [3] permettant d'effectuer en langage C lectures et/ou écritures d'un ou plusieurs bits ou mots vers l'automate, se traduisant in fine par les mêmes actions que précédemment. Le programme demandé doit créer un menu permettant d'effectuer au choix les différentes actions possibles. Certains étudiants vont même jusqu'à piloter un cycle complet de tri de colis.

2.2. Ressource IoT/Bdd

Pour le traitement des bases de données, le support choisi est le système de gestion de bases de données SQLite [4] (SGBD), notamment car ce type de SGBD est très répandu dans l'embarqué, et permet de bénéficier des avantages d'une base de données tout en gardant une simplicité de mise en œuvre (pas de serveur SQL à mettre en œuvre avec leurs problématiques de droits, gestion, etc.).

En TD, les étudiants apprennent tout d'abord à écrire des requêtes SQL dans le logiciel SQLitebrowser [5]. Ils peuvent utiliser des bases de données issues de l'open data [6] (comme les données de production électrique en France par exemple), ou créer leurs propres bases de données.

En TP, ils mettent en œuvre les requêtes SQL précédemment apprises, à l'aide d'un client écrit en C avec la bibliothèque SQLite. Dans le même esprit que dans la ressource précédente, le programme demandé doit créer un menu permettant d'effectuer au choix les différentes actions possibles (requêtes de lecture, d'écriture et de modification). Une seconde séance de TP est consacrée à une sensibilisation à Node-Red [7].

Node-Red est un outil de développement graphique/visuel ou *low-code* utilisant la notion de flux de données pour interconnecter les différents nœuds. L'édition des programmes se fait au travers d'un navigateur web[2]. Heureusement, ces blocs sont prédéfinis et il est rare d'avoir besoin de coder ses propres nodes. Il est à noter que les flux de données transitent au format JSON auquel une séance de TD est également consacrée.

Il est possible d'interconnecter une multitude de dispositifs : MQTT, base de données, I/O, Modbus, etc. Les étudiants sont donc amenés à effectuer des requêtes SQLite et MQTT et mettre en place des graphiques ou *dashboards* simples. MQTT est un protocole de messagerie $publish - subscribe$ basé sur le protocole TCP/IP, étudié également dans cette ressource, mais sortant du cadre de cet article.

Cette séance servira de base pour une des SAÉ.

2.3. Ressource programmation orientée objet C++/Qt (spé ESE et AII)

Cette dernière ressource a pour objectif d'initier les étudiants à la programmation orientée objets ainsi qu'à la création d'interfaces graphiques. Le langage C++ a été choisi, ainsi que le *framework* Qt et l'environnement de développement Qt Creator [8], afin de leur faire découvrir les possibilités de créations d'interfaces graphiques, souvent plébiscitées.

Les étudiants commencent par quelques TD/TP servant à appréhender les notions essentielles du C++ telles que, outre le principe $d'objets$, l'encapsulation, l'héritage et le polymorphisme. Des thématiques de GEII servent de base aux différents sujets, telles que les opérations entre nombres complexes et l'affichage de huit LED avec différents modes (chenillard, clignotement, compteur binaire) par exemple.

Notons que les bases de la programmation procédurale en langage C (instructions conditionnelles, boucles, appels de fonctions, pointeurs, ...) sont un socle non négligeable déjà appréhendées en première année (semestre 1) puis réutilisées dans le cadre de l'apprentissage de l'informatique embarquée, également en première année (semestre 2).

Suivent quelques TD/TP concernant les éléments de base des interfaces graphiques. De nouveau, on reste dans la thématique GEII avec une IHM permettant de calculer la valeur d'une résistance à partir des couleurs de ses trois bagues, puis une seconde IHM permettant de piloter l'affichage des huit LED vu en TD, en choisissant le mode d'affichage et différentes options (cf. Figure 4).

Cette compétence servira de base pour une autre des SAÉ.

[2] Les blocs/nodes sont principalement codés en JavaScript : la connaissance de ce langage est donc un atout

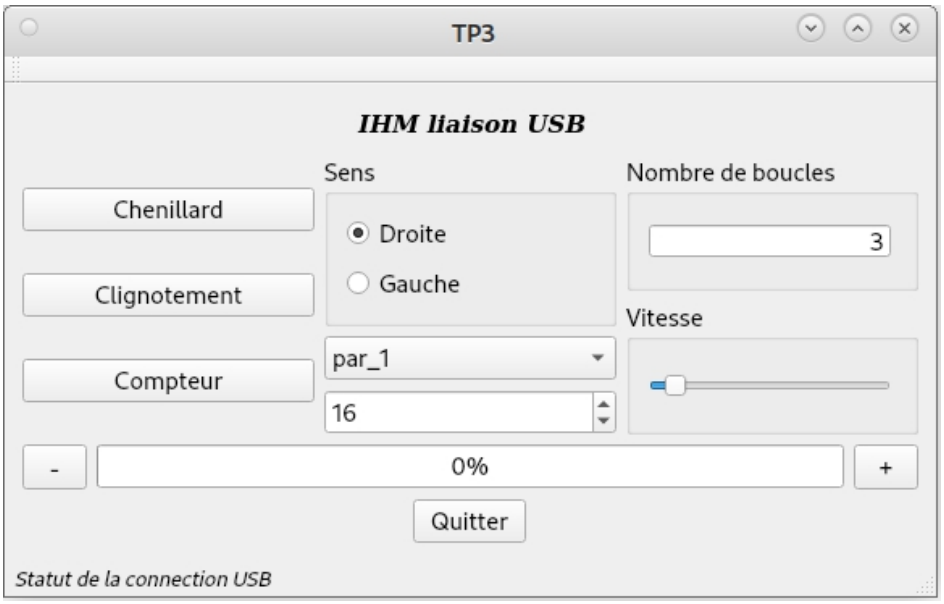


Figure 4. Exemple d'IHM réalisée en TP

3. SAÉ du parcours AII

Nous allons à présent présenter les trois sujets de SAÉ proposés aux étudiants de troisième année du parcours AII (semestre 5), à l'occasion desquels les étudiants sont amenés à utiliser les notions abordées dans les ressources présentées précédemment.

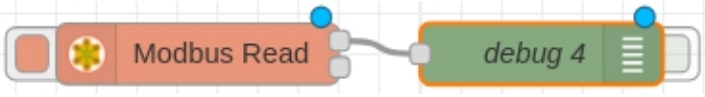
Il est à noter que dans les deux premiers sujets, le programme automate est fourni. Il s'agit du même programme que celui utilisé dans une ressource de supervision utilisant le logiciel PCVue. Dans une évolution future, les étudiants feront eux-mêmes le programme, comme dans le dernier sujet.

3.1. Programmation d'une interface graphique de pilotage/supervision d'un automate Schneider avec Node Red par protocole Modbus-TCP

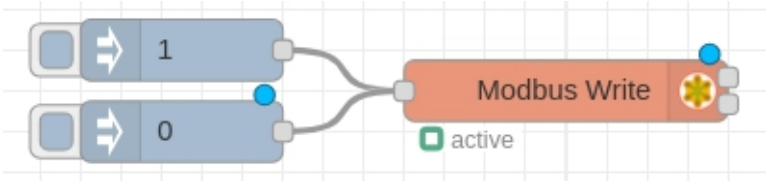
Il est demandé aux étudiants de créer une interface de pilotage/supervision de la maquette tri de colis déjà utilisée dans les ressources, accessible depuis un *dashboard web* réalisé avec Node-Red. La séquence se déroule sur environ 7 heures.

Le travail est décomposé comme suit :

- Rappels sur le protocole Modbus-TCP,
- Étude des entrées/sorties du programme de l'automate,
- Prise en main des blocs en lisant et écrivant des bits et mots (cf. Figure 5) ; la sortie se fait dans la fenêtre *debug*,



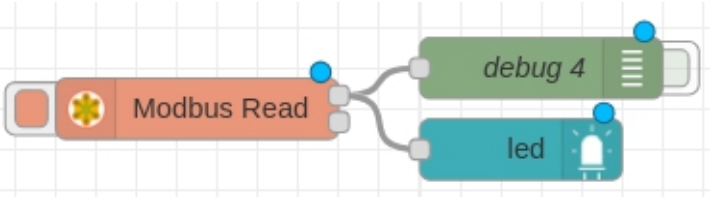
a) Bloc de lecture "Read Coil Status" avec sortie sur debug



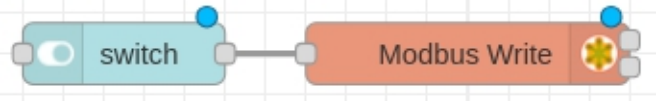
b) Bloc d'écriture "Force Single Coil" avec entrée 0 ou 1

Figure 5. Blocs de lecture/écriture de bits

- Affichage de l'état des entrées et pilotage des sorties depuis un *dashboard web* (cf. Figure 6),



a) Bloc de lecture "*Read Coil Status*" avec sortie sur un élément led du *dashboard*

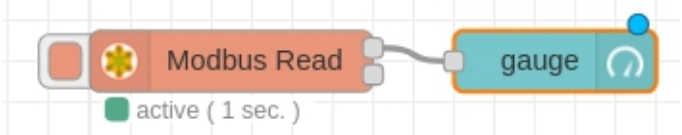


b) Bloc d'écriture "*Force Single Coil*" avec entrée depuis un élément *switch* du *dashboard*

Figure 6. Blocs de lecture/écriture utilisant un *dashboard*

- Ajout d'éléments de lecture ou d'écriture de mots avec un *slider* pour contrôler la vitesse du moteur et une gauge pour afficher sa valeur effective (cf. Figure 7).

a) Bloc d'écriture "*Preset Single Register*" avec entrée depuis un élément *slider*



b) Bloc de lecture "*Read Holding Registers*" avec sortie sur un élément *gauge*

Figure 7. Blocs de lecture/écriture utilisant un *slider* et une gauge

Il est nécessaire d'ajouter des blocs avec un peu de javascript pour séparer les flux ou les combiner dans un tableau (cf. Figure 8).

Notons qu'un objectif ici est d'essayer de produire le moins de code possible.

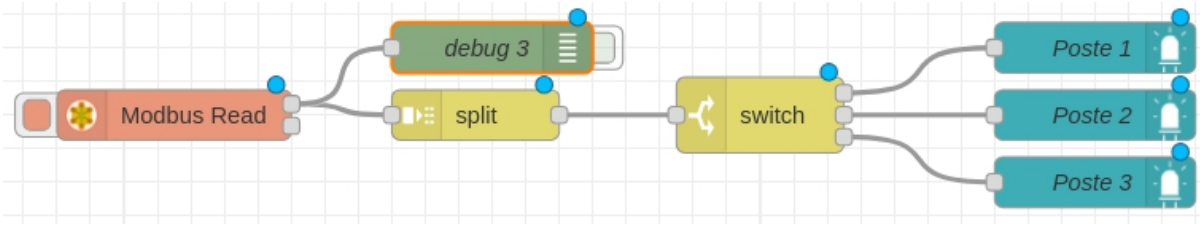


Figure 8. Blocs de lecture "*Read Coil Status*", lecture de plusieurs registres consécutifs

Par ailleurs, la solution retenue pour le split est de configurer des longueurs de 1 (cf. Figure 9), et pour le bloc switch de prendre les index `msg.parts.index` et de les affecter à chaque sortie (cf. Figure 10).

Figure 9. Configuration du bloc de séparation des flux

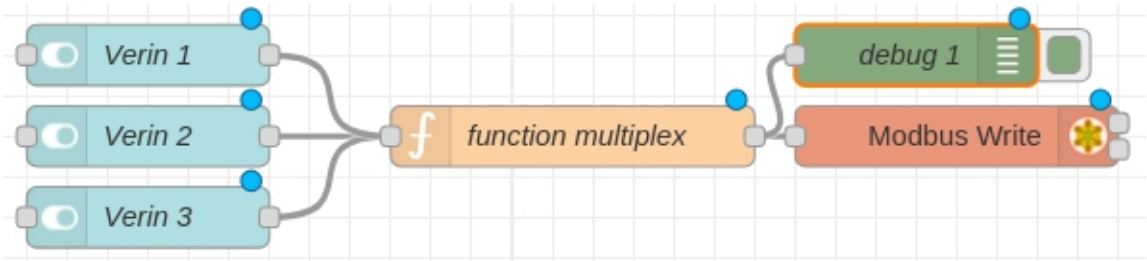


Figure 10. Blocs d'écriture "*Force Single Coil*", écriture de plusieurs registres consécutifs

Ci-dessous le code de la function multiplex de la Figure 10 :

```
flow.tab[msg.topic] = msg.payload;
msg.payload = flow.tab;
return msg;
```

3.2. Programmation d'une interface graphique de pilotage d'un automate Schneider avec le *framework* Qt par protocole Modbus-TCP et sauvegarde dans une base de données de type SQLite

Dans cette séquence d'environ 7 heures également, il est demandé aux étudiants de réaliser une application de bureau dont l'interface sera visuellement sensiblement la même que pour le sujet 3.1, mais développée en langage C++ avec le *framework* Qt.

Le travail est décomposé comme suit :

- Analyse des codes d'exemples de la documentation officielle Qt pour créer des fonctions de lecture et d'écriture par le protocole Modbus :
  - Association d'un bouton à l'écriture d'un bit,
  - Association d'un *slider* à l'écriture d'un mot,
  - Association d'un **QLabel** à l'affichage de l'état d'un bit lu,
  - Association d'un **QLCDNumber** à l'affichage de l'état d'un mot lu,
- Consolidation du code pour avoir des fonctions versatiles :
  - **writeRequest** prenant en paramètre le type de requête *Coils* ou **HoldingRegisters**, l'adresse de registre et la valeur à écrire. Cette fonction sera appelée par chaque bouton ou *slider*,
  - **readRequest** prenant en paramètre le type de requête *Coils* ou **HoldingRegisters** ainsi que l'adresse de registre. Les lectures se font registre par registre pour le moment,
- Mise en place d'un *scheduler/timer* pour mettre à jour à intervalle régulier tous les afficheurs. La période retenue est d'une seconde.

Enfin, il est demandé aux étudiants d'enregistrer les états des capteurs et les utilisations des actionneurs dans une base de données SQLite[3]. Ici la **libmodbus** est abandonnée au profit de **QSql**, la bibliothèque native de Qt. En suivant les programmes d'exemples et la documentation, les étudiants arrivent à transposer les notions vues en C vers le C++/Qt.

L'IHM finale est présentée Figure 11.

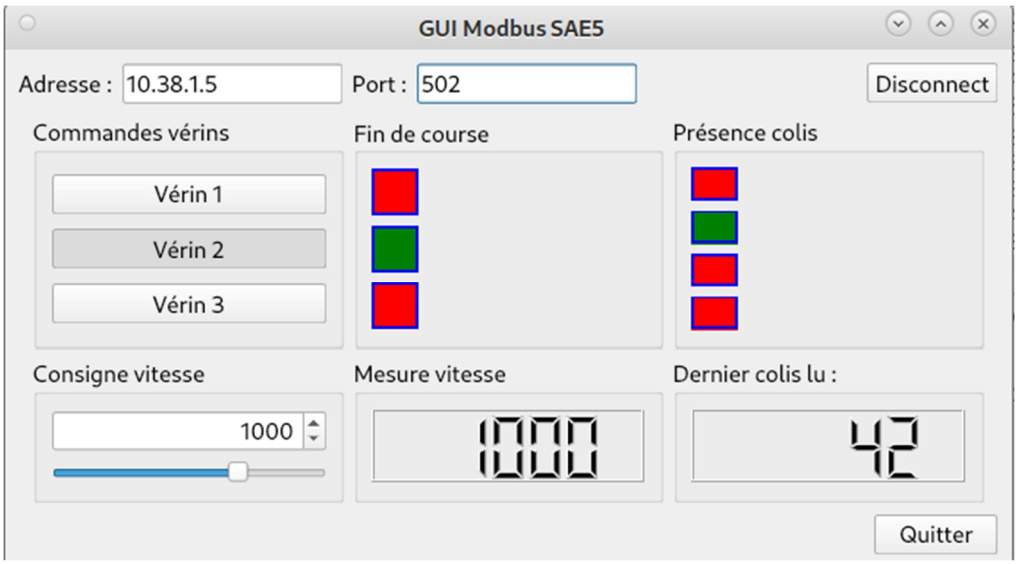


Figure 11. IHM de pilotage de l'automate

[3] Ceci permet de faire le lien avec la ressource Maintenance

3.3. Mise en œuvre d'une liaison Modbus RS485 entre un automate Schneider et un variateur de vitesse de moteur asynchrone

Cette séquence, d'une durée de 7 heures également, a pour but de mettre en œuvre une communication se rapprochant de ce qui se fait dans l'industrie. Nous allons ici aussi examiner la mise en œuvre d'une liaison Modbus RS485 entre un automate programmable industriel (API) Schneider et un variateur de vitesse Altivar pour moteur asynchrone (cf. Figure 12). L'objectif est de guider les étudiants dans chaque étape : depuis la préparation des matériels jusqu'aux tests de validation, en passant par la configuration des équipements et la programmation. Une attention particulière est accordée à leur capacité à trouver les informations pertinentes dans la documentation technique, un savoir-faire essentiel en milieu professionnel. En effet, aucune information autre que les documents techniques (choisis par l'enseignant) n'est donnée. Il s'agit d'un travail en totale autonomie.

Figure 12. Maquette moteur-variateur-automate

Figure 13. Variateur ATV312

Le travail est décomposé comme suit :

- Préparation de la liaison Modbus RS485 : Dans cette étape, les étudiants doivent eux-mêmes identifier les informations clés dans les documentations fournies. Par exemple, l'adresse des registres Modbus, les paramètres de communication coté automates, cotés variateur,
- Schéma de câblage : Les étudiants connectent les bornes A et B du port RS485 de l'automate aux bornes correspondantes du variateur (cf. Figure 13),
- Configuration des équipements :

  Automate M340 : En utilisant Unity Pro (ou Control Expert), les étudiants configurent le port Modbus avec les paramètres suivants : vitesse de transmission (19200 bauds), Format (8 bits, parité paire, 1 bit de stop (8E1)), Adresse (adresse unique, par exemple ADDM('0.0.0.@')),

  Les étudiants sont évalués sur leur capacité à documenter chaque étape de la configuration, C'est bien évidement utile pour s'y référer plus tard, mais aussi pour développer de bonnes pratiques de traçabilité,

- Programmation : Les étudiants utilisent les blocs fonctionnels WRITE_VAR et READ_VAR dans Unity Pro pour lire ou écrire sur les registres Modbus. Cette étape nécessite qu'ils identifient les registres pertinents à partir de la documentation Schneider. Par exemple, lecture : mot d'état (ETAD, adresse 8603), écriture : consigne de vitesse (LFRD, adresse 8602).
  Ils apprennent également à construire des programmes en LADDER ou en ST pour implémenter des séquences de commande, comme le démarrage ou l'arrêt progressif du moteur.
- Tests : Pour valider la communication, les étudiants mettent en place une table d'animation dans Unity Pro. Cette activité leur permet de visualiser les valeurs échangées en temps réel et diagnostiquer les erreurs de configuration en observant des anomalies dans les registres ou les messages d'erreur (par exemple, l'état NST sur le variateur).

Cette séquence pédagogique offre aux étudiants une expérience concrète des exigences de l'industrie. En les impliquant dans la recherche autonome des informations techniques, elle renforce leur capacité à résoudre des problèmes complexes et à s'adapter à divers contextes professionnels. Ce projet peut être adapté à différents niveaux de compétence et enrichi par des cas pratiques supplémentaires.

4. Conclusion

Nous venons de détailler l'utilisation du protocole Modbus comme socle de plusieurs SAÉ de troisième année du parcours AII en BUT GEII.

Nous avons tout d'abord détaillé l'apprentissage du protocole, ainsi que d'autres aspects nécessaires aux SAÉ, au travers du plusieurs ressources de deuxième et troisième année. Nous avons présenté ensuite les trois sujets d'applications du protocole Modbus, aux travers de mises en situation industrielles. Ces dernières, tout en possédant chacune une spécificité

propre, se veulent redondantes sur certains aspects, afin que les étudiants acquièrent des automatismes.

Les étudiants en apprentissage dans ce domaine sont satisfaits de se confronter à des applications mettant en œuvre un protocole, certes vieux, mais encore très utilisé dans leurs entreprises. Ils peuvent ainsi attester auprès des autres étudiants de la pertinence de ces choix d'applications. L'ensemble des étudiants trouve que ces applications leur ont permis de mieux appréhender les notions abordées en SAÉ au travers de cas concrets. Certains étudiants émettent un petit bémol quant au sujet basé sur de la programmation C++ car ce n'est pas leur cœur de métier, mais reconnaissent son intérêt dans le cadre de la formation GEII.

Dans les prochaines années, il est envisagé de placer la ressource C++/Qt en amont de celle sur les bases de données IoT pour n'utiliser que le C++/Qt. Pour l'instant le manque de disponibilité de certaines bibliothèques est un frein à cette organisation.

Par ailleurs il est envisagé de passer tout ou partie du parc automates de Schneider à Siemens. Ce sera l'occasion pour les étudiants de se confronter aux problématiques d'interopérabilité.

Références

[1] *DB Browser for SQLite [En ligne]. Disponible : https://sqlitebrowser.org. [consulté le 03 mars 2025].*

[2] *Wireshark [En ligne]. Disponible : https://www.wireshark.org. [consulté le 03 mars 2025].*

[3] *SQLite [En ligne]. Disponible : https://www.sqlite.org. [consulté le 03 mars 2025].*

[4] *Qt Creator [En ligne]. Disponible : https://www.qt.io/product/development-tools. [consulté le 29 avril 2025].*

[5] *pyModbusTCP 0.3.0 [En ligne]. Disponible : https://pypi.org/project/pyModbusTCP/. [consulté le 03 mars 2025].*

[6] *Open Data [En ligne]. Disponible : https://www.data.gouv.fr/fr/. [consulté le 06 mars 2025].*

[7] *Node-Red [En ligne]. Disponible : https://nodered.org. [consulté le 03 mars 2025].*

[8] *libmodbus 3.1.11 [En ligne]. Disponible : https://libmodbus.org/. [consulté le 03 mars 2025].*

[9] *Grafana [En ligne]. Disponible : https://grafana.com. [consulté le 03 mars 2025].*

Biographie des auteurs

Sophie DUPUIS est Maîtresse de Conférences au département Génie Electrique et Informatique Industrielle (GEII) de l'IUT de Montpellier et au Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM). Elle est en charge des enseignements d'électronique numérique, et informatique et réseaux.

Vincent THOMAS est professeur agrégé de Génie Electrique au département Génie Electrique et Informatique Industrielle (GEII) de l'IUT de Montpellier. Il est en charge des enseignements d'électronique, informatique et réseaux, et SAE.

Bertrand GELIS est professeur agrégé de Génie Electrique au département Génie Electrique et Informatique Industrielle (GEII) de l'IUT de Montpellier. Il est en charge des enseignements d'électrotechnique, informatique industrielle, automatismes, et SAE.