François-Xavier Cockenpot IMA4

Projet IMA4

Simulation d'un ascenseur en vue de sa commande par un automate réel

Sommaire

Introduction

Dans le cadre du deuxième semestre de quatrième année d'école d'ingénieur, il nous est proposé de réaliser un projet. Son but est de nous apprendre à réaliser un projet en mettant en oeuvre les connaissances acquises en IMA. Il permet de nous mettre en situation similaire à celle que nous allons rencontrer durant nos futurs stages et emplois.

La première étape fut de choisir le sujet. Etant orienter en système autonome dans la filière IMA, j'ai préféré me rapprocher des sujets d'automatiques. Un domaine m'a particulièrement attiré. Il allie à la fois le côté automatisme que je recherche mais demande aussi de travailler sur la communication réseau. J'ai choisi de réaliser le projet de simulation d'une ascenseur en vue de sa commande par un automate réel via une liaison série. Le protocole de communication utilisé sera le Modbus.

Ce projet me permet de travailler sur un Automate Programmable Industriel (API) mais aussi d'utiliser un logiciel de simulation tout en ayant un aspect réseau.

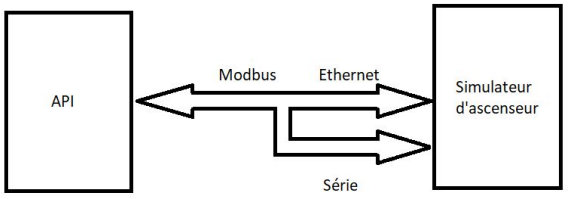
I Présentation du projet

Dans cette partie, je vais vous présenter de façon générale mon projet et expliquer les différentes solutions choisie

1) Contexte du projet

Créé en 1979 par modicon, le protocole Modbus est aujourd'hui encore grandement utilisé en industrie. N'ayant pas eu l'occasion de travailler sur ce protocole en 4ème année, j'ai choisi ce sujet dans le but de découvrir et d'acquérir de nouvelles compétences. J'utiliserai donc le Modbus dans le but de contrôler un ascenseur simulé sur ordinateur depuis un API.

2) Composition du système



Dans ce projet, on trouvera différentes parties.

Il y aura une partie simulation que j'effectuerai sur le logiciel Labview 2013. On y trouvera donc la simulation de l'ascenseur. Cette partie ne devra avoir aucun contrôle sur l'ascenseur puisque celui ci doit être entièrement contrôlé par l'automate.
Sur l'écran de simulation, on pourra observer l'ascenseur en mouvement, l'état de ses capteurs, de ses actionneurs et les boutons d'appels.

La seconde partie est la partie contrôle du simulateur. Elle sera faite sur le logiciel Unity pro. Ce logiciel a été créé par Schneider Automation. Il permettra donc de programmer toute la partie contrôle dans différents types de langage : Grafcet et Ladder. Le programme sera directement implanté dans l'API.

La dernière partie est donc la partie réseau. Le protocole Modbus sera utilisé. Ce protocole fonctionne sur le mode Maître - Esclave. Le maître écrit et lit dans chaque esclave. C'est lui qui a la main sur tout le réseau.
Dans le but de réaliser le projet le plus représentatif de la réalité, l'automate sera considéré comme maître et l'ascenseur comme esclave.
Cette partie est donc présente dans les deux premières parties puisque le réseau est présent du côté simulateur comme du côté automate.

3) Cahier des charges

Nous allons fixer ici les limites du projet.

On se limitera ici à un ascenseur 2 étages, typiquement un ascenseur présent dans un immeuble d'habitation, avec une priorité descendante. Pour expliquer brièvement cette priorité, si l'ascenseur est appelé à l'étage 1 et 2 au même moment, il ira d'abord au 2ème étage puis s'arrêtera au 1er pour repartir au rez-de-chaussée par exemple.
Une communication entre l'automate et le simulateur sera mis en place et la communication sera continue et instantanée.

L'automate enverra des requêtes de lecture et d'écriture sur les variables du simulateur et celui-ci lui répondra.

Les capteurs présents sur le simulateur sont les capteurs d'étage, de limite haute et basse et d'état de la porte. Des boutons poussoirs seront présents pour faire les appels d'ascenseur.

Des leds seront présentes pour afficher les ordres reçu par le simulateur (monter - descendre - ouvrir porte - fermer porte).

II. Réalisation du projet

Dans cette partie, je traiterai de la réalisation du projet. J'expliquerai les solutions retenues et les problèmes rencontrés. La première partie portera sur le simulateur, le seconde sur l'automate et la troisième parlera de la partie Modbus.

1) L'ascenseur simulé sur Labview

Je vais vous expliquer ici la démarche que j'ai suivi pour réaliser la simulation de l'ascenseur le logiciel Labview. Pour commencer, un des pré-requis de ce projet était de travailler sur une version de Labview récente comparé à la version utilisée actuellement dans les tp de 4ème année. La programmation de l'ascenseur sera repris futurement dans les tp de réseau industriel de 4ème année d'ima. La première étape fut d'installer une nouvelle version sur les ordinateurs.

Une fois ce pré-requis réalisé, j'ai pu commencer à faire l'interface utilisateur. Le premier choix que j'ai du réaliser été de trouver une solution pour créer un ascenseur sur le logiciel. N'étant pas de base spécialisée pour créer des objets tel un ascenseur, il a fallu trouver une solution. Après quelques recherches sur le logiciel, la solution que j'ai retenu a été l'utilisation de barre de progression horizontale. Deux barres sont nécessaires pour simuler les deux portes. Ces barres peuvent être contrôlés en valeur et en position. En effet, pour simuler un ascenseur, il faut bien évidemment que l'objet créé soit mobile pour simuler la montée et la descente de celui ci, mais aussi gérer l'ouverture et la fermeture des portes. J'ai utilisé un outil qui permet d'agir sur les propriétés d'un objet, le property node. Il a alors suffit de gérer l'incrémentation sur la propriété de position top de celui ci lorsque l'événement montée ou descente à lieu ou sur la propriété de valeur lorsque l'évènement ouverture ou fermeture des portes.

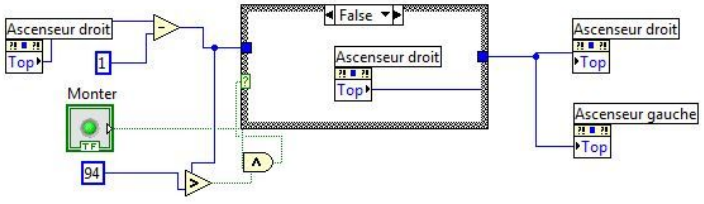


Figure 1 : Gestion montée ascenseur

Sur la partie Labview, nous avons aussi besoin d'avoir des capteurs de position. Il est impossible de créer un capteur qui détecte un objet passant devant lui. J'ai donc décidé de scruter la valeur de la propriété de position top sur les réservoirs. Il suffit de scruter différentes valeurs (ici 500 pour niveau rez-de-chaussée - 300 pour étage 1 - 100 pour étage 2 et de prendre des valeurs légèrement supérieures et inférieures pour les limites hautes et basses de l'ascenseur). Ces capteurs sont représentés par des leds sur le simulateur.

Une fois les différents capteurs mis en place, reste à gérer l'appel palier. Dans un ascenseur réel, les boutons poussoirs s'allument lorsque nous appuyons dessus. Il est ici impossible d'allumer les boutons poussoirs pour signaler un appel puisque ceci devrait être géré en entrée et en sortie par l'automate. On utilise des boutons bascules (qui reste à 1 le temps qu'on reste appuyé dessus) pour créer l'appel et des leds gérées par l'automate pour signaler que l'appel a bien été détecté.

J'ai aussi ajouté 4 voyants permettant de voir les ordres reçus pour pouvoir s'assurer du bon fonctionnement de l'automate.

2) Programmation de l'automate sur Unity Pro Xl

Toute la partie contrôle doit avoir lieu sur un API. En collaboration avec mes tuteurs, j'ai choisi de travailler sur les automate Schneider présent dans une salle de tp de polytech. La programmation sur ces automates est réalisée sur le logiciel Unity Pro Xl.

Tout d'abord, pour pouvoir réaliser le projet à bien, on a bien entendu besoin d'un module Modbus sur l'automate. Ce module sera relié à l'ordinateur via un cable RS-282 puis un adaptateur RS-282 / USB à l'ordinateur.

Pour pouvoir vérifier cette partie, j'avais obligatoirement besoin d'une partie qui arrivera par la suite, la communication Modbus entre l'automate et le simulateur. On considérera ici que nous avons accès aux différentes variables mais aussi la possibilité de contrôler l'ascenseur avec les requêtes d'écriture.

La première partie est de créer le programme principal de contrôle. Le programme est réalisé en Grafcet et toutes les actions sont écrites en ladder. Celui ci gère la montée et la descente de l'ascenseur mais a aussi la possibilité de déclencher le grafcet de gestion des portes. Pour faire ceci, nous avons besoin d'avoir des variables en commun. Le programme principal déclenche le programme secondaire en activant des variables booléens et attend lui même que ce deuxième active sa variable de fin de programme. Ces variables sont ensuite reset.

Le programme principal doit prendre en compte tous les cas possibles. En effet, si l'ascenseur, par exemple, s'arrête trop tard suite à un problème réseau, il doit être capable et retourner de lui même à un étage. C'est aussi lui qui gère le fonctionnement de l'ascenseur, c'est à dire, selon l'état des différentes variables d'entrées, savoir à quel moment déclencher chaque action.

En parallèle de ces deux programmes, deux autres grafcets tournent en continue. Ces deux programmes servent à la gestion des boutons. Le premier détecte chaque front montant sur les boutons poussoirs du simulateur permettant de gérer des variables internes à l'automate. A chaque détection de front montant, des variables sont activées. Le deuxième est responsable de l'état des led d'appel ascenseur sur le simulateur. Il fait appel aux variables de position puisque les voyants activés dans le programme précédent sont reset ici lorsque l'ascenseur est arrivé à l'étage voulu. Par exemple, Le voyant de l'appel étage 2 extérieur sera éteint lorsque les variables de position étage 2 et des portes ouvertes de l'ascenseur seront vérifiées.

3) La communication Modbus

Je vais aborder ici la communication Modbus ayant lieu entre l'automate et le simulateur d'ascenseur. Cette partie sera séparée en deux parties puisque le réseau a besoin d'être géré des deux côtés. La partie automate sera considérée comme le maître sur le réseau et le simulateur comme l'esclave. Cela signifie que l'automate envoie toutes les requêtes. Le simulateur, lorsque les ordres lui sont adressés, doit réagir et répondre. En effet, avec le protocole Modbus, il n'y a qu'un seul maître mais il peut y avoir une multitude d'esclaves. Généralement, les esclaves sont eux aussi des automates, dont l'adresse peut être gérée facilement avec deux boutons tournants. Comme nous utilisons ici un simulateur, je prendrai en compte uniquement les requêtes ayant pour identifiant l'adresse esclave deux.

Pour commencer, je vais vous présenter les trames émisent par l'automate. C'est ici du Série Modbus. Des fonctions Read_var et Write_var sont présentes dans le logiciel Unity Pro Xl. Elles permettent de lire et d'écrire sur les variables du simulateur si les fonctions sont bien paramétrées.

La requête Read_var envoit ce type de trame :

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Slave address (ici 2)	Function code (ici 0x03)	Starting address high	Starting address low	Number of points high	Number of points low	Error check	Error check

La requête Write_var envoit ce type de trame :

Field name	Example	RTU	ASCII	
Start of frame	-	t1-t2-t3	":"	0x3a
Slave address	0x0B	0x0B	"0B"	0x30, 0x42
Function code	0x10	0x10	"10"	0x31, 0x30
Starting address high	0x00	0x00	"00"	0x30, 0x30
Starting address low	0x00	0x00	"00"	0x30, 0x30
Number of register high	0x00	0x00	"00"	0x30, 0x30
Number of register low	0x02	0x02	"02"	0x30, 0x32
Byte Counter	0x04	0x04	"04"	0x30, 0x34
Data high (register 0)	0x12	0x12	"12"	0x31, 0x32
Data low (register 0)	0x34	0x34	"34"	0x33, 0x34
Data high (register 1)	0x56	0x56	"56"	0x35, 0x36
Data low (register 1)	0x78	0x78	"78"	0x37, 0x38
Error Check (LRC / CRC)	-	0xA9 0x43	"CB"	0x43, 0x42
End of frame		t1-t2-t3	-	0xD, 0xA

Table 5.46: Example inquiry, Preset Multiple Registers

Figure 2 : Requête d'écriture

J'ai pu détecter le format des trames en mettant une sonde sur la liaison série dans la partie Labview

Dans l'élément Adr, on rentre l'adresse dans laquelle nous voulons lire les variables, on met ici ADDM('0.0.0.2'). ADDM permet de convertir, ce qu'on met en paramètre, en code pour la fonction. Les 3 premiers 0 sont l'adresse du rack du processeur, l'adresse du modicon dans le rack et le numéro de port série qui sont toujours 0 dans ce cas. Le 2 représente le numéro de l'esclave. Ensuite, nous avons le type d'objet à lire, ici des mots mémoires. L'indice du premier mot à lire et le nombre d'élément à lire ne sont pas ici très important puisque l'on ne s'adresse pas à un automate mais à un simulateur. Une table de gestion est nécessaire pour vérifier le bon fonctionnement de la fonction. On récupère alors la réponse dans le tableau Recp qui est un tableau de 2 entiers. On convertit alors les 2 entiers en 2 tableaux de 16 bits permettant d'avoir accès aux valeurs de chaque variable. Une fois cela réalisé, on a alors directement accès au valeur des capteurs et des boutons poussoirs.

Pour le write_var, la fonction est quasiment identique, cependant on n'a ici plus de reception, mais une valeur émise lors de la réalisation de la fonction. De même que pour la partie précédente, ce qui est émit est un tableau d'entiers, on convertit donc un tableau de bit correspondant aux ordres émis et aux valeurs des voyants d'appel étage dans un tableau

d'entiers. On a ici un tableau de dimension 2. J'ai séparé les ordres ascenseurs et les voyants pour plus de clarté.

Une fois cette fonction bien contrôlée, il suffit de les insérer dans un grafcet cyclique continue. Lorsqu'une requête est envoyée, la fonction attend une réponse. Ceci est géré dans la partie simulateur.

Dans la partie simulation, la première étape est d'initialiser la liaison série. Une librairie liaison VISA est disponible dans Labview, elle permet de paramétrer la liaison série assez facilement en prenant en compte la parité, le vitesse de transmission etc. Lors du lancement du programme Labview, j'ai mis une phase ou le port série est d'abord vidé, puis fermé. Je ré-initialise ensuite le port série avec mes paramètres et j'ouvre ensuite la liaison.
A partir de ce moment la, on entre dans une boucle while qui attend l'appuie sur le bouton stop. Dans cette boucle, on peut, grâce à une fonction dans Labview, scruter le nombre d'octets présents dans le buffer du port série. Une fois 8 octets détectés dans le buffer du port série, on lit le port série. Deux cas sont alors possibles :

- La requête émise par l'automate est un Read_var. La longueur de la requête est 8 octets. On peut savoir que c'est une lecture en regardant la valeur de l'octet 1 qui vaut 3. On peut alors directement créer une réponse avec les paramètres constants de la réponse, les 3 premiers octets, puis y ajouter les octets correspondants au valeur des capteurs et des boutons poussoirs (transformé depuis un tableau de booléen vers un entier), puis en y ajoutant le CRC calculé selon le début du message. On utilise alors la fonction d'envoie de message sur le port série.

- La requête émise par l'automate est un Write_Var. La longueur de la requête est 13 octets. On détecte l'écriture par une valeur de l'octet 1 de 16. On doit continuer à scruter le port série pour attendre les 5 octets manquants au message. Une fois les 5 octets dans le buffer du port série, on complète le message lu précedemment. 4 octets correspondent au valeur des variables envoyées. On convertit alors les entiers en tableaux de booléen et on écrit les valeurs dans les différents variables. Une réponse est ensuite envoyée à l'automate. Elle est tout le temps identique et est juste utile à savoir que le message a bien été transmis et reçu.

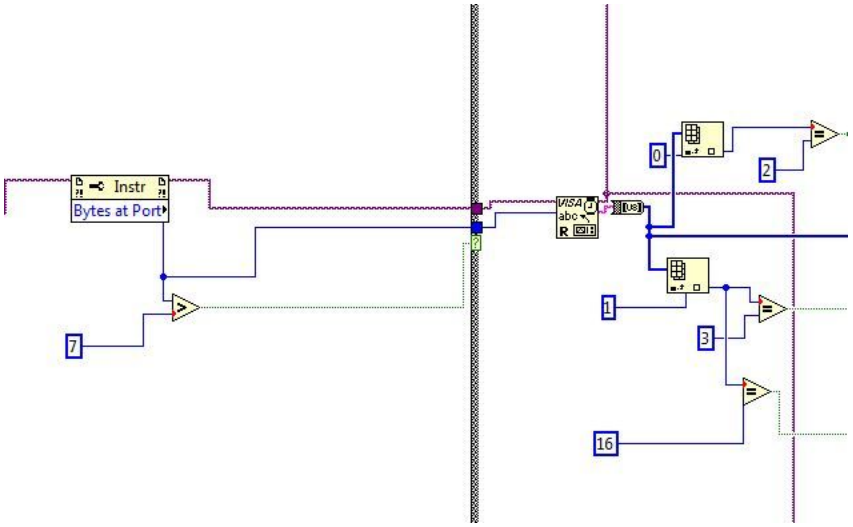


Figure 3 : Lecture trame

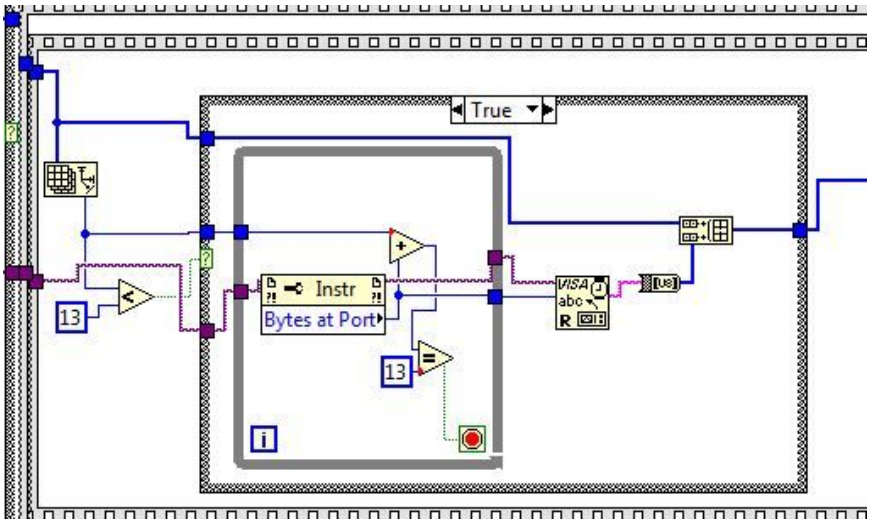


Figure 4 : Gestion cas lecture

Une fois tout cela mis en place, la liaison Modbus est opérationnelle. L'automate peut aisément avoir accès aux valeurs de capteurs, aux boutons poussoirs mais on peut aussi contrôler l'ascenseur et les valeurs des voyants d'appels.

III. Résultats atteints, problèmes rencontrés et perspective d'évolution

1) Résultats atteints

Le résultat final obtenu répond au cahier des charges initial. Sur le simulateur, on peut observer l'ascenseur agir selon la volonté de l'automate tout en observant l'état des différents capteurs, actionneurs et voyants. L'automate contrôle en temps réel l'ascenseur grâce au réseau Modbus.

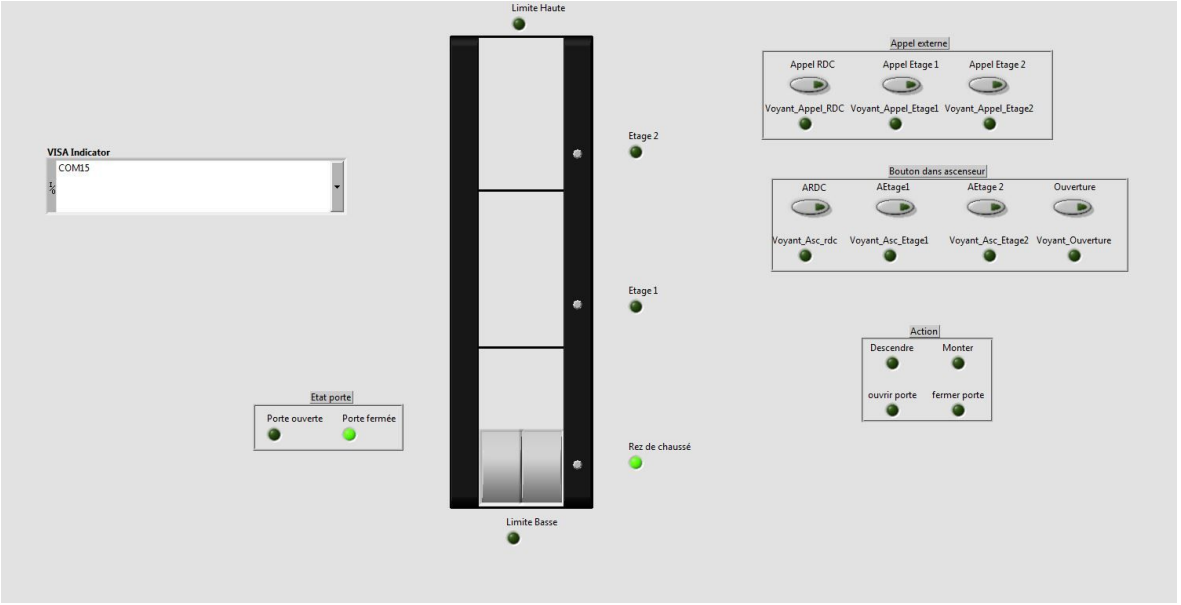


Figure 5 : Interface finale

2) Problèmes rencontrés

Durant tout le projet, j'ai pu noter quelques problèmes majeurs.

Tout d'abord, lors de la réalisation de la partie réseau sur Labview, le problème rencontré fut que la liaison série, bien que initialisée et n'affichant aucune erreur ne marchait pas. Lorsque je mettais des sondes sur les sorties de lecture de port série, rien n'apparaissait. Après des heures de recherche sur ce problème, j'ai désinstallé la version antérieure de Labview présent sur le pc. J'avais installé la version 2013 de Labview mais en laissant la version antérieure, un conflit sur le driver de la liaison série avait lieu, et le labview 2013 utilisait alors une version de driver non compatible. Ne pouvant pas avoir 2 versions de driver différentes sur la même pc, la désinstallation du Labview antérieure était nécessaire.

Le second problème important rencontré porte aussi sur la liaison série. Je n'ai pas résolu ce problème mais je pense avoir détecté la raison. Lorsque Labview est lancé, si la ram est saturée, le port série affiche une erreur. J'ai pu tester ça en lançant un programme gourmand en ressource et cela faisait que cette erreur devenait récurrente. En temps normal, cette erreur peut très bien ne jamais apparaître.

3) Perspective d'évolution

Une des possibilité d'évolution de ce projet est d'utiliser le logiciel Labview 2013 pour les tp de réseau industriel en IMA4. Il serait alors possible de travailler sur le protocole Modbus en commandant l'ascenseur avec un PC. On aurait donc un PC de contrôle et un PC avec le simulateur d'ascenseur, les deux reliés par un module WAGO. Ces deux modules reliés ensemble, chaque pc serait considéré comme le maître et le module WAGO comme esclave. On utiliserait sur un pc une liaison série Modbus et sur l'autre PC une liaison Ethernet Modbus.
Ce genre de tp est aujourd'hui réalisé sur une version de Labview antérieure et sur un contrôle non pas d'ascenseur mais de cuve. Il serait donc bien de pouvoir évoluer vers un tp utilisant un logiciel plus récent et susceptible d'être utilisé en entreprise.

Conclusion

Durant ce projet, j'ai pu travaillé sur différents aspects du programme étudié pendant notre cursus en IMA. Ce projet m'a permi de travailler sur de l'automatisme tout en ayant une partie réseau. J'ai aussi pu découvrir d'autres aspects de logiciel. En effet, lorsqu'on utilise Labview pendant nos tp, nous partons d'un programme déjà fait que nous complétons légèrement. J'ai ici pu réaliser entièrement le programme et bien comprendre le fonctionnement ce celui-ci.

De plus, devoir mettre en réseau deux parties bien distinctes m'a fait progresser dans le fait de bien poser les choses, permettant de récupérer les informations voulus facilement et clairement.

Ce projet est très intéressant dans le fait que c'était le premier projet où on reliait un pc à un automate via un protocole Modbus et que celui-ci est une réussite.

Bibliographie

https://www.schneider-electric.fr/fr/faqs/FA147283/

https://fr.wikipedia.org/wiki/Modbus

http://zone.ni.com/reference/fr-XX/help/371361K-0114/lvinstio/visa_open/