Table des matières

1 Introduction

Le projet MOOLs (Massive Open Online Laboratories) propose d'étendre le principe de MOOCs aux travaux pratiques. Il propose de réunir tous les outils techniques utiles à la commande à distance et à la supervision des infrastructures de laboratoire. Une application HTML dédiée à chaque expérimentation et intégrée dans un navigateur Web permet aux utilisateurs d'interagir avec l'expérience de laboratoire par l'intermédiaire de panneaux de commande et de supervision. Il est indispensable que l'étudiant puisse visualiser l'expérience de laboratoire à l'aide de signaux et d'images réelles (par retour vidéo), d'une part pour connaître en permanence l'état du système, d'autre part pour donner une impression de réalité, malgré la distance. Les outils de développement «National Instruments» et LabVIEW sont utilisés pour le développement de l'architecture serveur-client ainsi que pour le transfert bidirectionnel des données entre les clients et les page Web de manipulation.

Les MOOCs permettent aux étudiants d'assister à un cours en ligne à travers n'importe quel navigateur internet. Un des avantages de ce type de cours est le nombre théoriquement illimité de personnes suivant le cours.
Ce principe peut être appliqué aux laboratoires. Au lieu de participer à un cours, l'étudiant peut se connecter à un des laboratoires mis en ligne et interagir, à travers une application, avec les installations des différents travaux pratiques. Le laboratoire peut être effectué à n'importe quel instant de la journée. L'application propose de sélectionner plusieurs maquettes (Régulation de vitesse ou de position d'un moteur DC, Régulation de position de bille) et de sélectionner différents types de régulations (boucle-ouverte, PID, PI, P, P + Feed-Forward etc.).

Le but de ces expériences est l'application et l'expérimentation à distance des bases théoriques de régulation acquises par les étudiants.

Ce projet de diplôme s'intègre à un grand projet de recherche financé par SUDAC (Swissuniversities Development and Cooperation Network), dont le but est de mettre les laboratoires des partenaires Suisses (EPFL et HES-SO Valais) à disposition des pays en voie de développement (Iran, Niger, Lebanon et Djibouti).

2 Cahier des charges

Le cahier des charges fourni avec le sujet du travail de diplôme contient les quatre points suivants :

- Conception et mise en œuvre d'une plateforme avec plusieurs expériences de laboratoire (Régulation d'un moteur DC, Mesure et commande des panneaux solaires orientables, etc.). Chaque expérience de laboratoire est équipée de différents dispositifs de commande et de mesure ainsi que d'une Webcam.
- Développement d'une infrastructure informatique articulée autour d'une architecture client-serveur.
- Développement des interfaces utilisateurs avec des panneaux de commande et de supervision.
- Tests, vérifications expérimentales et optimisations.

3 Description générale de l'infrastructure des MOOLs

L'installation peut être "découpée" en quatre couches distinctes. Les éléments qui les composent sont les suivants :

- Smart Devices + Lab Servers
- Web Server
- Moodle
- Client

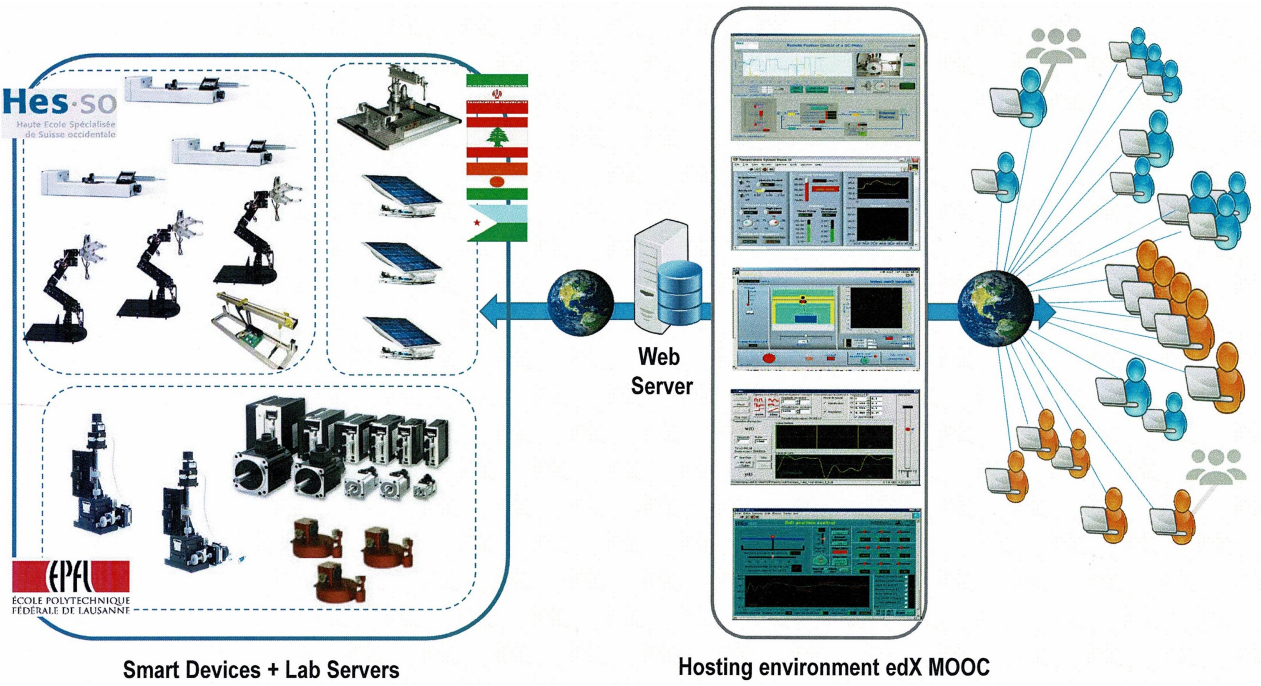


FIGURE 3.1 – Schéma de principe des "MOOLs"

3.1 Web Server

L'intelligence étant intégrée dans les Smart-Devices, le serveur n'a pour rôle que de relayer les informations entre ces dernières et les différents clients.
Il doit cependant être capable de gérer le fait que plusieurs clients se connectent en même temps au même appareil. Ainsi, les clients doivent être mis dans des files d'attente jusqu'à la libération d'un des postes. Afin d'éviter qu'un client bloque les autres, un temps maximal est fixé pour le pilotage d'une maquette. De ce fait, après ce temps limite, le premier client de la file est automatiquement déconnecté du laboratoire. S'il souhaite reprendre le contrôle d'une des maquettes de l'infrastructure, il doit simplement recharger la page web de contrôle. Il sera cependant mis en dernière position de la file d'attente.

3.2 Smart Devices

Le terme de "Smart Devices" comprend les maquettes à piloter et leur régulateur connecté à internet. Dans le cadre de ce travail de diplôme, il s'agit d'une régulation de vitesse et de position d'un entraînement électrique ainsi que la régulation en position d'une bille sur un rail. A long terme, d'autres expériences seront ajoutées à l'infrastructure.

L'entraînement électrique est composé d'un moteur DC, de deux capteurs de position (codeur incrémental et potentiomètre) ainsi que d'un amplificateur de puissance pour alimenter le moteur.

La maquette de régulation de position de bille est composée d'un servomoteur, d'un capteur de position optique et d'un amplificateur de puissance pour le moteur.

Ces deux installations sont pilotées par un automate industriel National Instrument cRIO-9067. Les différents signaux de commandes et de mesures des installations sont gérés par les cartes d'entrées/sorties analogiques et numériques de l'automate. Ces différents signaux sont commandés ou visualisés par le client à l'aide d'une page web programmée en HTML5.

Chaque "Smart Device" possède également une webcam connectée en USB à l'automate qui permet de rajouter du réalisme au laboratoire distant. L'image de cette caméra est envoyée sur la page web du client.

3.3 Moodle

Les étudiants qui souhaitent se connecter à l'une des installations se connectent sur la plateforme Moodle du laboratoire. Ils peuvent ainsi charger la page HTML de contrôle de l'installation dédiée à l'expérience. Le cours en ligne possède différentes sections pour chaque expérience. Ces sections hébergent les différentes pages Web des expériences.

3.4 Clients

Les clients peuvent être classés en deux catégories, les observateurs et les contrôleurs. Les observateurs ne peuvent qu'observer l'installation à distance contrairement aux contrôleurs qui peuvent la piloter.
Comme évoqué précédemment, les clients sont placés dans des files d'attentes. Ce qui définit le type de client (observateur ou contrôleur) est simplement la position de ce dernier dans la file d'attente. Le premier de la file est le seul contrôleur de l'installation et tous les clients le suivant sont classés comme observateur.
Cette file est organisée selon le principe du "premier arrivé, premier servi". Si un client décide par lui-même de se déconnecter du laboratoire, sa place est prise par le client qui s'est connecté juste après lui.

4 Environnement de développement LabVIEW

Les Smart-Devices ainsi que le Web-Serveur sont programmés en LabVIEW.
LabVIEW est un logiciel conçu à l'origine pour de l'acquisition de données sur la plate-forme Macintosh. Il a par la suite évolué pour piloter des automates, des FPGAs et autres instruments d'entrées sorties. Ses domaines d'applications sont, entre-autre l'acquisition de données, l'automatisation, la supervision de processus et l'enseignement.

La programmation est de type graphique et peut être exécutée sur un ordinateur sous la forme d'un exécutable grâce à l'"Application Builder", ou sur des appareils d'entrées/sorties National Instrument. La programmation est faite dans des Instruments Virtuels, abrégés "VI". Ceux-ci contiennent d'une part une interface utilisateur (Front Panel) permettant d'interagir avec le code lorsqu'il est exécuté à l'aide de contrôles et d'indicateurs. D'autre part, ils contiennent un écran de programmation (Block Diagram) qui comprend le code sous la forme de composants câblés.

5 Protocole de communication Smart-Devices - Client

Le contrôle des différentes expériences se faisant via une interface web, la communication entre le client et le Smart Devices est basée sur un protocole IP (Internet Protocol).

« *Internet est un réseau de réseaux développé par le ministère de la Défense aux États-Unis dans la fin des années 70 et le début des années 80 pour interconnecter les différentes machines informatiques de ce ministère. La solution a été de développer un protocole commun que l'ensemble des réseaux et des machines connectées doit posséder. Ce protocole commun, c'est précisément le protocole IP (Internet Protocol). Les réseaux interconnectés, que nous avons appelés les sous-réseaux, peuvent être quelconques. Il leur est juste demandé de transporter d'une extrémité à l'autre des paquets IP, c'est-à-dire des paquets conformes aux spécifications du protocole IP.*» [4]

Pour que cette communication virtuelle puisse avoir lieu, les deux appareils doivent parler le même langage et suivre les mêmes règles. Pour ce faire, chaque appareil possède des composants qui implémentent ces différentes règles plus communément appelées protocoles. Un appareil qui possède plusieurs composants peut donc communiquer avec un autre appareil sous différents protocoles. Les protocoles HTTP, TCP, UDP et IP font partie des plus utilisés. LabVIEW, qui peut être considéré comme un composant, est capable de communiquer avec, entre autres, les protocoles précédemment cités. Il s'agit donc ici de déterminer quel est le meilleur protocole à adopter dans le cas des laboratoires à distance.

5.1 Architecture de la communication

La communication peut être séparée en deux parties. Il y a tout d'abord une communication entre les différents "Smart-Devices" et le serveur, puis une autre communication entre le serveur et les différents clients. Le Web-Server ainsi que les Smart-Devices sont programmés en LabVIEW. Côté client, la programmation est exclusivement faite en HTML5. Le schéma suivant représente les canaux de transmission des données entre les différents acteurs du MOOLs.

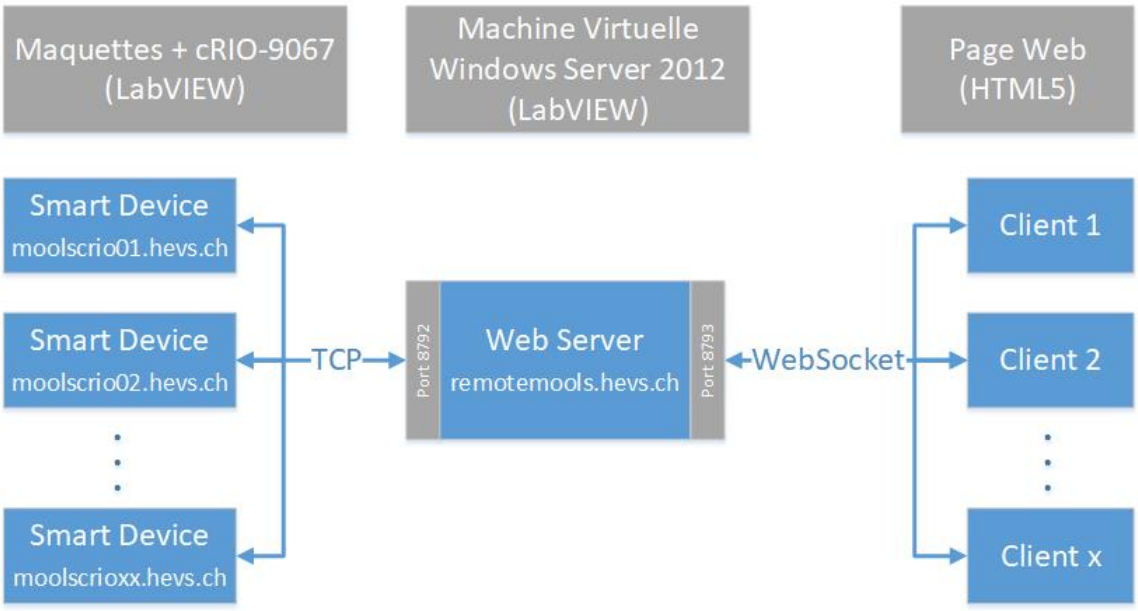


FIGURE 5.1 – Architecture du réseau MOOLs

La machine virtuelle permet aux clients d'accéder depuis l'extérieur aux Smart-Devices qui sont dans l'infrastructure réseau de la HES-SO Valais à Sion. Pour des questions de sécurité, les automates cRIO ne peuvent pas être visibles depuis un réseau externe à l'école. La solution proposée par le service informatique de l'école est de passer par une machine virtuelle. Celle-ci est accessible depuis l'extérieure par le port TCP 8793 à l'adresse "remotemools.hevs.ch". Cette adresse est le nom DNS publique visible depuis l'extérieur donné à la machine pour les MOOLs.

En interne, la machine virtuelle accède aux automates via le port TCP 8792. L'adresse IP de ces automates est fixe et dépend de leur emplacement physique dans l'infrastructure de l'école. Dans le but de simplifier la programmation, un nom DNS leur est attribué et a la forme suivante : "moolscrioxx.hevs.ch". Comme il est prévu de connecter plusieurs automates au laboratoire, deux caractères sont réservés ("xx") pour le numéro de l'automate.

Pour que cette communication soit possible, le service informatique a ajouté des règles aux Firewalls situés entre les automates et le serveur, ainsi qu'entre le serveur et l'extérieur.

5.2 Communication Smart-Devices - Serveur

Plusieurs types de communications peuvent être implémentés entre les deux applications LabVIEW du SmartDevice et du serveur. Il s'agit ici de déterminer lequel des protocoles de communication est le plus adapté pour cette application. Pour ce faire, différents protocoles sont implémentés est comparés.

5.2.1 TCP

Le premier protocole à faire l'objet d'une étude est le TCP/IP. Son implémentation est native à LabVIEW et est relativement simple à mettre en place.

« Le protocole TCP (Transmission Control Protocol) a été initialement défini dans la RFC 793 de l'IETF (Internet Engineering Task Force).
Le protocole TCP fournit un service de transport fiable, basé sur la connexion et le "bytestream", en plus des connexions non fiables service réseau fourni par IP. TCP est employé par un grand nombre d'applications comme par exemple :

- *Email (SMTP, POP, IMAP)*
- *World Wide Web (HTTP, ...)*

Sur internet, la plupart des applications utilises le protocole TCP. De nombreuses études ont rapportées que TCP était responsable de plus de 90% des données échangées sur Internet.» [1]

La forme d'un paquet TCP est la suivante :

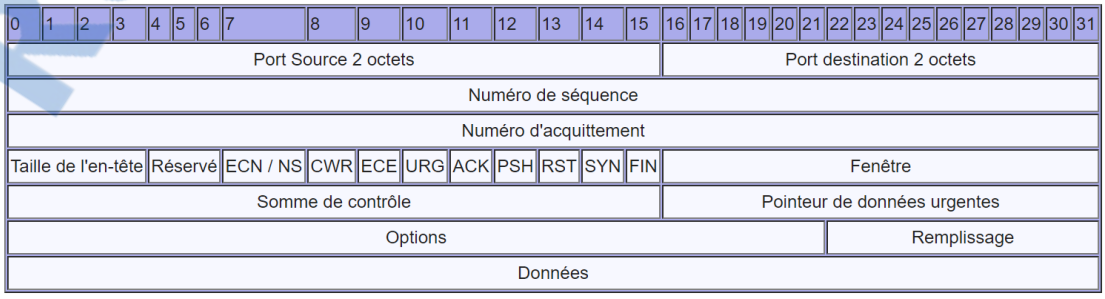


FIGURE 5.2 – TCP Header
Source : `https://fr.wikipedia.org/wiki/Transmission_Control_Protocol` (consulté le 01.08.2018)

Dans LabVIEW, la mise en forme du paquet est gérée directement par la bibliothèque TCP. Il suffit ainsi de donner en argument de la fonction "TCP Open Connection" le numéro de port ainsi que l'adresse IP du destinataire. Les données sont fournies en argument du bloc fonction "TCP Write" sous la forme d'une chaîne de caractères.

La figure suivante représente la structure du programme utilisé afin de comparer le TCP aux autres standards.

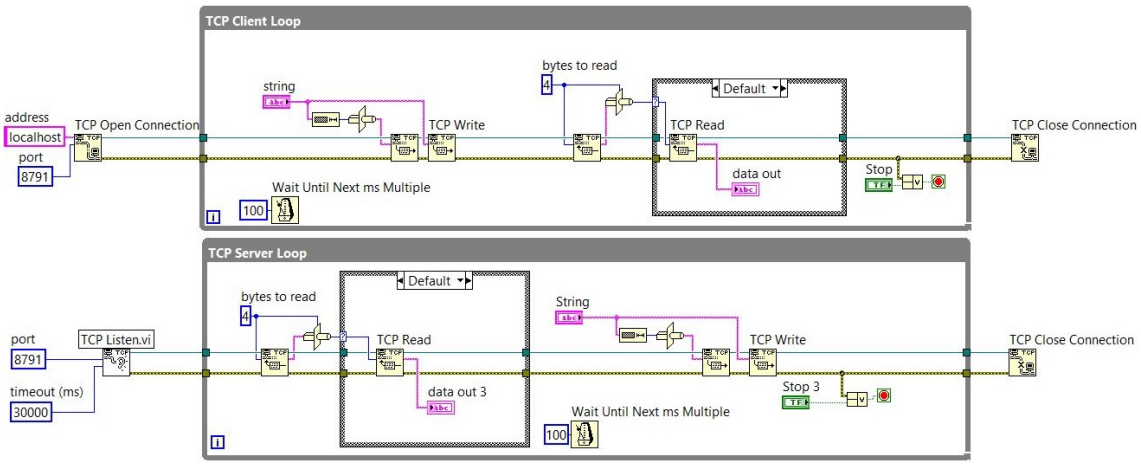


FIGURE 5.3 – TCP Read/Write

En LabVIEW, la structure de programmation d'une communication est souvent la même. A l'initialisation du programme, une connexion (TCP ou autre) est ouverte côté client. Côté serveur, le programme écoute les requêtes entrantes. La boucle While du serveur n'est ainsi démarrée que lorsqu'une requête arrive sur le port d'écoute. Le temps entre les paquets transmis entre le serveur et le client est défini par le cadencement des boucles While. Cet intervalle est fixé par le bloc fonction «Wait Until Next ms Multiple». Pour éviter la congestion des données, les deux boucles doivent respectivement écrire et lire les messages à la même fréquence.

Les performances du protocole TCP sont analysées et comparées aux autres protocoles dans le chapitre 5.2.4.

5.2.2 DataSocket

Le deuxième protocole à faire l'objet d'une étude est le "DataSocket" qui est un protocole propriétaire. Il a été développé par National Instrument et n'est utilisé que par leurs produits.
Il est implémenté par-dessus une couche TCP et son but est de simplifier la programmation pour l'échange de données entre appareils National Instrument. Le DataSocket permet d'échanger n'importe quel type de données LabVIEW entre deux composants NI, contrairement au TCP qui ne peut envoyer qu'une chaîne de caractère. Les données n'ont ainsi pas besoin d'être converties et mises sous la forme d'une chaîne de caractère, ce qui facilite grandement la programmation. Cependant, un désavantage du DataSocket est le fait que l'on ne puisse pas choisir le port TCP utilisé. Il est en effet fixé à 3015. Ceci peut être problématique si le Firewall, que ce soit du côté client ou serveur, bloque ce port et si on ne peut pas l'autoriser pour une raison quelconque.

La structure de programmation Serveur-Client est la suivante. Le principe reste le même que pour le TCP.

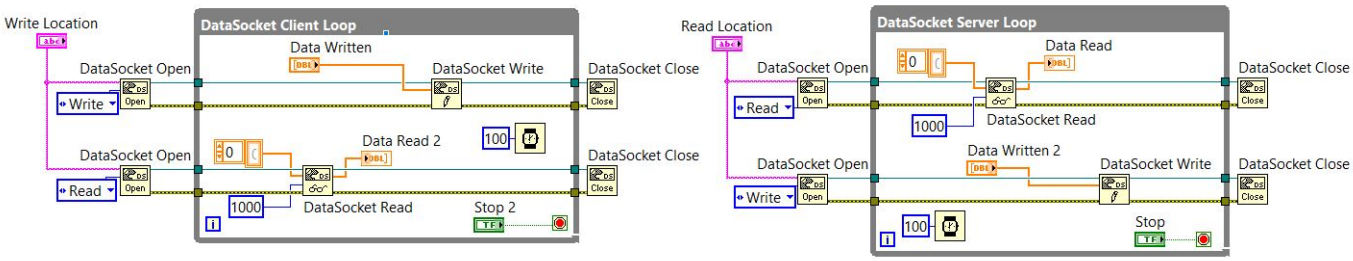


FIGURE 5.4 – DataSocket Read/Write

Avec le DataSocket, une communication doit être ouverte pour chaque variable transmise. Ainsi, si on souhaite envoyer 10 variables par exemple, 10 communications doivent être ouvertes en parallèle. Un moyen de contourner cela est d'encapsuler les différentes variables dans un Cluster. Un Cluster est une structure de donnée qui permet de regrouper plusieurs données de même ou de différents types dans une seule variable.

5.2.3 UDP

Le dernier protocole étudié est l'UDP.
« *Le protocole UDP (User Datagram Protocol) est défini dans la RFC 768 de l'IETF. Les principales caractéristiques du service UDP sont les suivantes :*

- *le service UDP ne peut pas fournir d'unités SDU (Service Data Unit) dont la taille est supérieure à 65507 octets*
- *le service UDP ne livrera pas un SDU corrompu à la destination*
- *le service UDP ne garantit pas la livraison des unités SDU (des pertes peuvent se produire)*

»[1]
Les pertes intrinsèques au protocole UDP sont problématiques dans le cas de la transmission de signaux temporels ainsi que de signaux de commandes. Ce protocole n'est donc pas une bonne solution pour piloter et visualiser les SmartDevices. Cependant, pour la transmission de l'image, il n'est pas problématique si de temps à autres l'image reçue est corrompue. Il est ainsi envisageable d'utiliser le protocole UDP seulement pour la transmission de l'image.

Son implémentation dans LabVIEW est la même que pour le TCP. Les blocs TCP de la figure 5.3 sont simplement remplacés par les blocs UDP.

« La grande majorité du trafic TCP/IP est de type TCP (environ 90 UDP est utilisé principalement dans les contextes suivants :

- *Echange de données court (peu de petits paquets) : la simplicité de UDP est appréciée dans ce cas.*
- *Communication « temps réel » : parfois des données correctes reçues trop tard n'ont pas d'utilité. C'est le cas par exemple dans des applications nécessitant un comportement temps réel comme le « streaming audio ou vidéo » ou la téléphonie sur IP.*

Le tableau ci-dessous résume les principales différences entre UPD et TCP :

TABLE 1 – Comparaison TCP et UDP

TCP	***UDP***
TCP est « orienté connexion » (connection oriented).	*UDP est « non orienté connexion » (connectionless) : chaque paquet est autonome et il n'y a pas de notion de connexion.*
TCP offre un service de transmission fiable de type stream.	*UDP considère des paquets de manière individuelle. Aucune gestion d'erreur n'est incluse.*

[2] »

5.2.4 Comparaison des ressources et choix du protocole

Afin de déterminer le meilleur protocole pour cette application, il est judicieux de comparer les ressources réseaux que demandent ces différents standards. Pour ce faire, l'utilisation de la carte réseau du PC jouant le rôle de serveur est mesurée à l'aide du moniteur de ressources Windows. Dans un premier temps, la mesure est effectuée à deux reprises pour la transmission de signaux. La première avec une simple communication TCP et la seconde avec l'ajout d'une couche DataSocket. Aucune mesure n'est faite avec le protocole UDP dans cette première partie pour les raisons évoquées dans le chapitre 5.2.3.

Pour comparer les protocoles, deux applications LabVIEW (client et serveur) sont implémentées, une fois en TCP et une autre en DataSocket. Le client envoie au serveur, à titre d'exemple, les trois paramètres d'un PID sous la forme de variable de type "double" (nombre réel codée sur 32 bits) ainsi que les paramètres de forme (sinus, square, etc.), d'amplitudes, de fréquence et d'offset d'un signal temporel périodique. La taille d'un paquet fait ainsi 208 bits et le temps entre deux messages est fixé à 100ms. Dans le cas du TCP, ces données sont converties sous forme de "String" avant d'être envoyées.

Concernant le serveur, il envoie au client deux signaux temporels produits par un générateur de fonctions. Ces deux signaux sont échantillonnés à une fréquence de 10kHz est sont envoyés par paquets de 1000 échantillons toutes les 100ms, soit des messages de 80 MBytes/s.

La figure 5.5 représente l'utilisation de la carte réseau du serveur pour le protocole TCP et DataSocket.

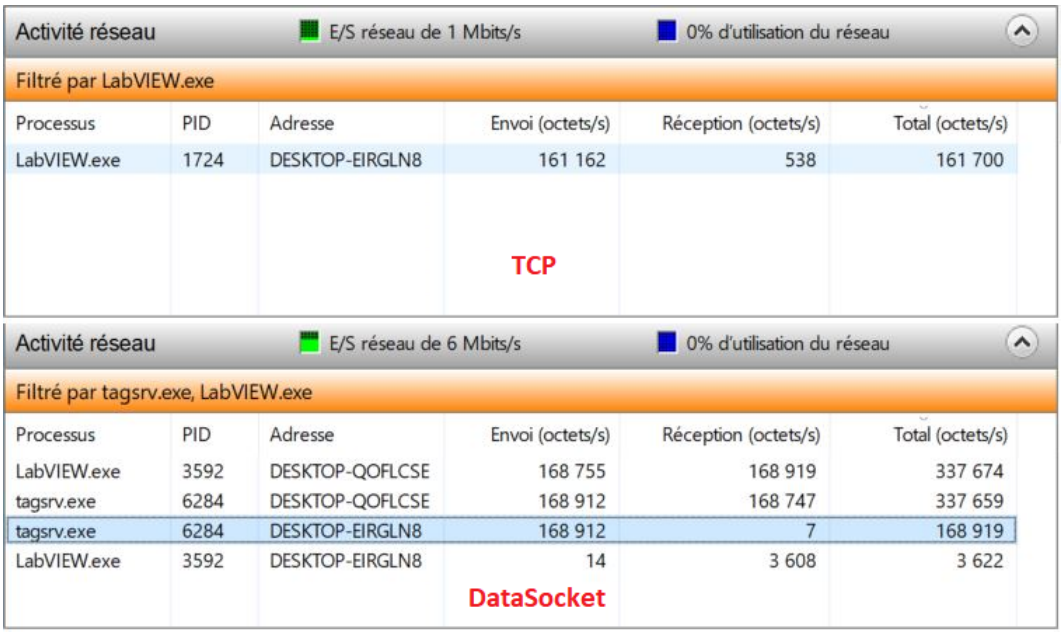


FIGURE 5.5 – Utilisation de la carte réseau sans caméra (TCP vs DataSocket)

L'adresse "DESKTOP-EIRGLN8" est celle du serveur et "DESKTOP-QOFLCSE" celle du client. Il est ainsi possible de déterminer dans quelle direction vont les paquets.
Comme le montre les deux moniteurs ci-dessus, la différence de ressources entre les deux protocoles est faible. Le protocole TCP envoie 161'162 bytes/s au client alors que le protocole DataSocket envoie 168'926 bytes/s.
Au niveau de la réception, la différence est un peu plus flagrante, le TCP reçoit 538 bytes/s et le DataSocket 3615 bytes/s.

Dans le cas du DataSocket, on remarque sur les deux premières lignes du tableau qu'il échange des données en local. Ceci est certainement dû au fait que LabVIEW transfère dans un premier temps les données à l'exécutable "tagsrv.exe" avant de les envoyer au client.

Dans un deuxième temps, l'image de la webcam du serveur est envoyée au client en plus des données précédentes. L'image possède une résolution de 1280x720 pixels et est envoyée toute les 100ms.

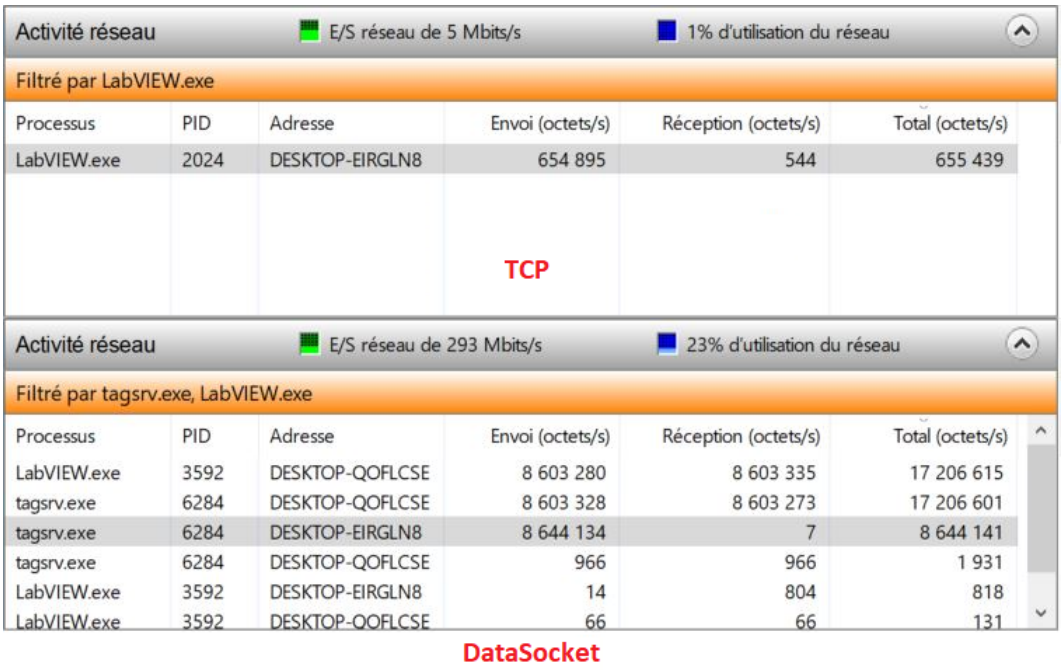


FIGURE 5.6 – Utilisation de la carte réseau avec caméra (TCP vs DataSocket)

Cette fois-ci, la différence est beaucoup plus importante entre les deux protocoles. L'image reçue par le client possède pourtant la même résolution et la même fréquence de rafraichissement dans les deux cas. Le protocole TCP n'envoie que 654'895 bytes/s alors que le DataSocket envoie 8'644'148 bytes/s, soit plus de 10 fois plus.

Finalement, un dernier test est effectué avec le protocole UDP pour la transmission de la vidéo.

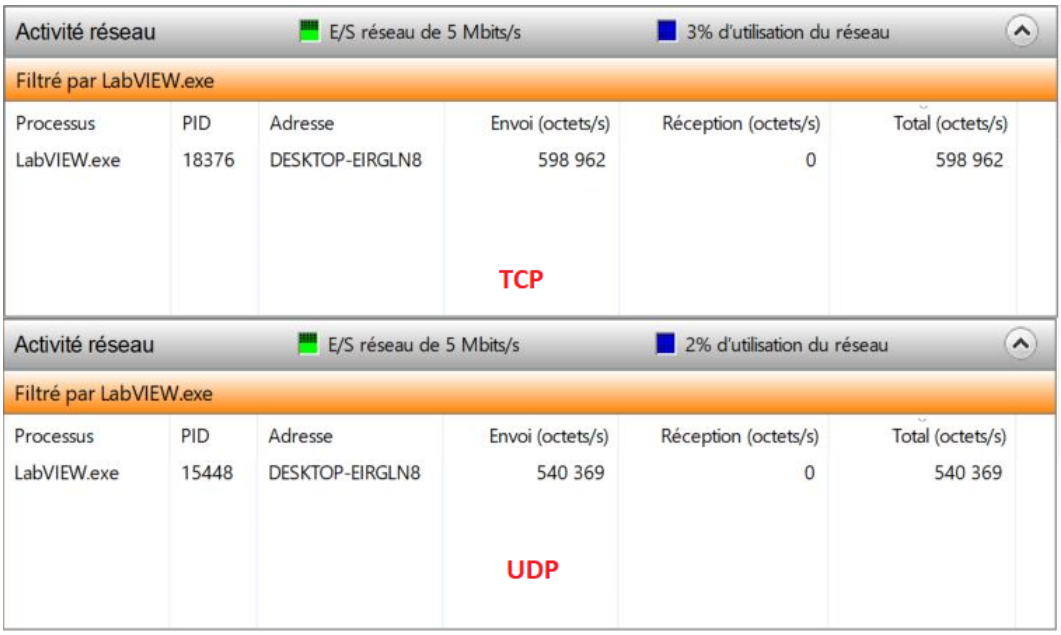


FIGURE 5.7 – Utilisation de la carte réseau avec caméra (TCP vs UDP)

La différence entre les deux protocoles est relativement faible (9.8%) Les images arrivent au client sans latence dans les deux cas et les paquets envoyés font à peu de chose près la même taille.

D'après ces différentes mesures, les protocoles TCP et Datasocket se valent pour la transmission de signaux. Concernant la transmission d'image, le DataSocket est de loin le pire des trois protocoles.

D'après ces résultats, le protocole choisi est le TCP. Il est privilégié pour éviter de programmer plusieurs protocoles différents en parallèle et d'ainsi compliquer le code. Au niveau des performances, c'est également le plus polyvalent dans le cas de ces laboratoires à distance.

5.3 Communication Serveur - Client

La communication serveur-client se situe entre la machine virtuelle et la page Web de contrôle (voir fig. 5.1). Le protocole de communication entre ces deux parties ne peut être le même que celui utilisé entre le serveur et les Smart-Devices, etant donné que de l'interface client programmée en HTML5 n'intègre pas directement le protocole TCP "brut". C'est pour cette raison qu'une couche additionnelle (WebSocket) est implémentée par-dessus le TCP.

5.3.1 WebSocket

Le protocole WebSocket est celui employé pour les laboratoires distants de l'EPFL. Il est donc judicieux de l'employer plutôt qu'un autre afin de se standardiser avec le travail déjà effectué par l'EPFL et ainsi de faciliter la mise en commun des ressources entre cette école et la HES-SO Valais.

Le WebSocket, qui est un standard du Web, a été normalisé en 2011. Il est développé par-dessus une connexion TCP.

« *Le protocole WebSocket permet une communication bidirectionnelle entre un client exécutant du code non fiable dans un environnement contrôlé vers un hôte distant qui a opté pour les communications de ce code. Le model de sécurité utilisé pour cela est le modèle "origin-based security model" couramment utilisé par les navigateurs Web. Le protocole envoi tout d'abord un "handshake" (établissement d'une liaison) suivi d'un message de base, superposé sur TCP. L'objectif de cette technologie est de fournir un mécanisme pour navigateur basé sur les applications qui nécessitent une communication bidirectionnelle avec des serveurs qui ne compte pas sur l'ouverture de plusieurs connexions HTTP.* » [3]

Dans le cadre des MOOLs, l'intérêt principal de ce protocole est le fait que le client n'a pas besoin d'envoyer une requête au serveur pour recevoir un paquet de données. En d'autres termes, après avoir établi la communication avec le serveur, le client peut recevoir les données du "Smart-Device" sans envoyer une requête pour chaque mesure ou chaque image. Ce type de communication est appelée "Real-time". La figure 5.8 représente ce type d'échange de données en comparaison avec des échanges traditionnels.

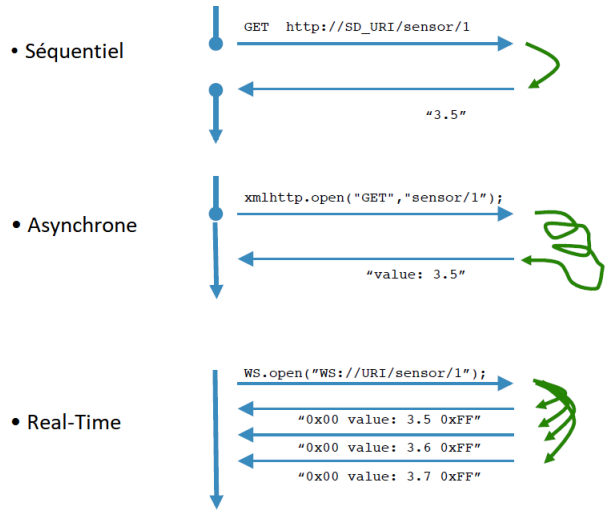


FIGURE 5.8 – Modèles de transmission [1]

1. Source : NIDays Smart device for remote Experimentation, Dr. Christophe Salzmann

Un autre avantage de ce protocole est le fait qu'il soit directement intégré à HTML5.
L'application du client peut ainsi être développée en HTML5 et être plus facilement intégrée à une page web existante qui serait hébergée dans le cours en ligne du laboratoire. Le client doit simplement ouvrir une page web pour accéder aux infrastructures du MOOLs et n'a pas besoin de télécharger d'application. Cela permet également une compatibilité maximale entre avec le différents systèmes d'exploitation. La seule condition requise est d'avoir un navigateur web à jour.

Pour les raisons évoquées en début de chapitre, ce protocole est choisi pour la communication entre le serveur et le client et aucun autre protocole n'est étudié.

5.3.2 WebSocket et LabVIEW

Le protocole WebSocket n'est pas nativement intégré à LabVIEW. Les paquets doivent par conséquent être mis en forme avant d'être envoyés via une connexion TCP. Pour la réception, ils doivent également être décryptés pour pouvoir exploiter leurs données. De plus, à l'initialisation de la communication, une clé doit être échangée entre le serveur et le client pour que la communication puisse avoir lieu.
Ces différentes fonctionnalités ont été développées à l'EPFL dans le cadre de leurs MOOLs. Ce projet se faisant en collaboration avec eux, leur développement LabVIEW sur le WebSocket nous a été fourni. La figure suivante représente le principe de programmation de la communication côté serveur.

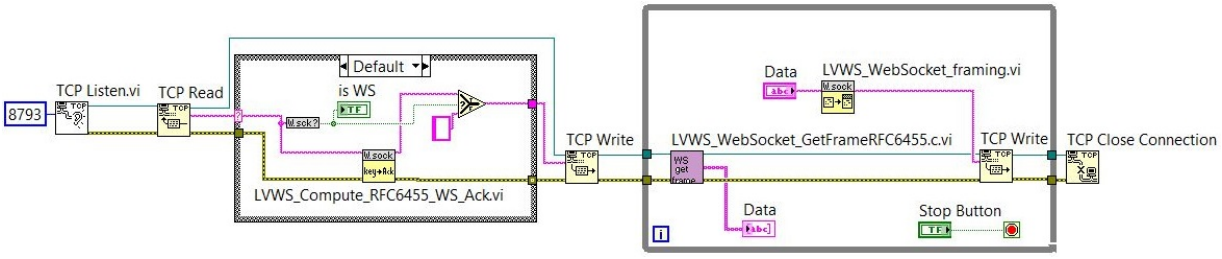


FIGURE 5.9 – Principe de communication WebSocket LabVIEW

L'établissement de la communication « Handshake » se fait à l'aide du VI "LVWS_Compute _RFC6455_WS_Ack". Si la communication entrante est bien basée sur du WebSocket, le VI renvoie une chaîne de caractères contenant la clé permettant d'accepter la communication. Par la suite, cette chaîne doit être renvoyée au client via TCP à l'aide du bloque "TCP write". Une fois cette procédure effectuée, l'échange de données entre le client est le serveur peut débuter.

Pour échanger des données entre le serveur et le client, deux VI sont nécessaires.
Pour la réception, le VI "LVWS_WebSocket_GetFrameRFC6455" est utilisé. Sa fonction et de décrypter le paquet WebSocket selon le standard RFC 6455 (voir figure 5.10). Il retourne les "Data" du message sous la forme d'une chaîne de caractères.

Pour l'envoi, c'est le VI "LVWS_WebSocket_framing" qui est utilisé. Sa fonction est de mettre en forme une chaîne de caractères toujours selon le standard RFC 6455. Ce VI ne fait que transformer les données. Il faut par conséquent ajouter le bloc "TCP Write" en série pour envoyer les données au client.

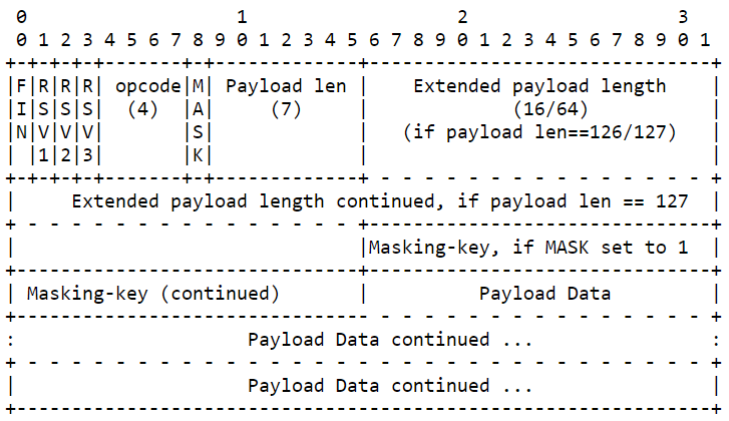


FIGURE 5.10 – RFC 6455
Source : https ://tools.ietf.org/html/rfc6455 (consulté le 01.08.2018)

5.3.3 WebSocket et HTML5

L'implémentation du WebSocket en HTML5 est relativement simple puisqu'il est nativement intégré. Son développement se fait entièrement dans le JavaScript de la page web. Pour créer une communication WebSocket, il faut tout d'abord créer un objet "WebSocket". Le constructeur "WebSocket" prend comme paramètre l'url de la connexion. Cette url doit avoir la forme suivante : "ws ://ipAddress".

A l'ouverture de la page web, deux communications sont ouvertes en parallèle, une pour les commandes et les mesures de la maquette (data), et une autre pour la vidéo. Pour les différencier au niveau du serveur, deux urls différentes sont utilisées. La première communication est initialisée avec l'url "ws ://remotemools.hevs.ch/ws1" et la vidéo avec l'url "ws ://remotemools.hevs.ch/ws2".

Les méthodes de l'objet "WebSocket" employées pour la communication sont les suivantes :

- **onopen** : fonction appelée à l'initialisation de la communication
- **onmessage** : fonction appelée à chaque réception de données
- **send** : fonction permettant d'envoyer les données
- **onclose** : fonction appelée à l'arrêt de la communication

Le schéma suivant décrit le fonctionnement de la communication.

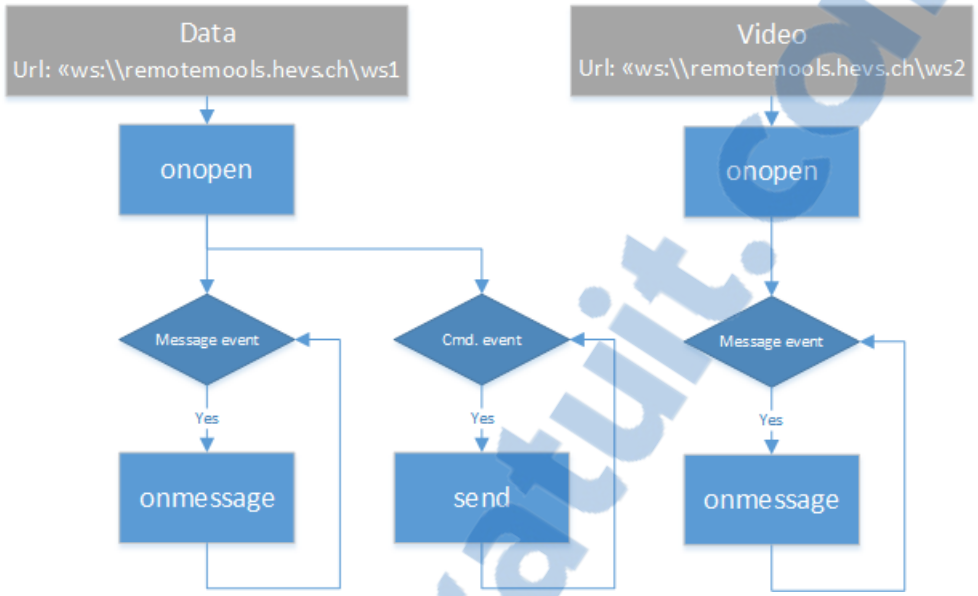


FIGURE 5.11 – Diagramme d'état de la communication WebSocket

A l'ouverture de la communication "ws1", la méthode "onopen" envoie une requête pour connaître la position du client dans la file d'attente, ainsi que son rôle (contrôleur ou observateur). Un second message est également envoyé pour demander au serveur de transmettre les différentes mesures.
En ce qui concerne la communication ws2, une seule requête est envoyée pour demander au serveur de transmettre l'image.
La fonction "onmessage" n'est appelée que lorsqu'un message est reçu. Ainsi, les mesures et l'image sont rafraîchies en fonction de la fréquence d'envoi des données du WebServer et donc en fonction du cadencement de la boucle d'envoi LabVIEW.

Les différents paramètres des Smart-Devices leurs sont également envoyés par évènement. Lorsque l'utilisateur change une valeur de commandes sur la page web, la fonction "send" est appelée. Elle renvoie alors au WebServeur tous les paramètres du Smart-Device.

L'implémentation de cette partie est détaillée dans le chapitre 9.3.

6 Web Serveur

Le Web Serveur est implémenté en LabVIEW sur une machine virtuelle tournant sous Windows Server 2012. Il joue le rôle de passerelle entre les Smart-Devices et les clients. Son but est donc de rediriger les différents paquets de données entre les automates cRIO et les pages web de contrôle. Il gère également les files d'attentes et le temps de contrôle limité des clients.

6.1 Gestion des connexions client

La structure de base utilisée pour ouvrir une connexion avec une page web est celle du chapitre 5.3.2. Si on emploie cette structure telle quelle, on ne peut communiquer qu'avec un seul client. Il faut donc complexifier quelque peu cette structure. Une solution pour pallier ce problème est d'utiliser une programmation avec lancement dynamique de VIs. Ce type de programmation permet d'exécuter des clones d'un même VI en parallèle. L'évènement qui déclenche le démarrage d'un de ces clones est la requête d'ouverture de communication du client. Comme décrit dans le chapitre 5.3.3, la page Web ouvre deux connexions à son ouverture. Cela fait par conséquent deux évènements et donc deux ouvertures de clones par client.

La figure 6.1 représente la programmation pour effectuer un lancement dynamique d'un VI.

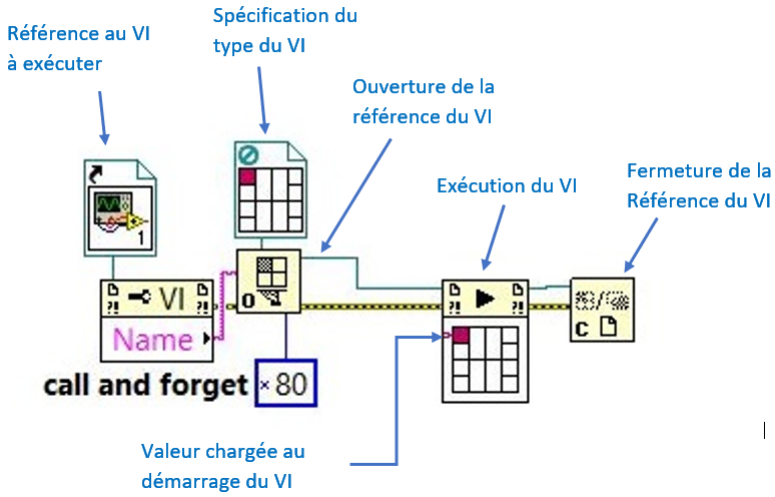


FIGURE 6.1 – Exécution dynamique d'un VI

Pour générer les évènements qui lance dynamiquement le VI, la bibliothèque "Queue Operations" est utilisée avec un "Listner" TCP.
En LabVIEW, le système de "queue" permet de sauvegarder dans des buffers n'importe quel type de données. Cela permet de sauvegarder des données dans un premier temps, et de les traiter par la suite. Cette structure de programmation suit le principe de "producteur/consommateur". Le producteur ajoute les données dans la "queue" lors d'un évènement et le producteur va lire ces données en les retirant une à une de la "queue".

La figure 6.2 représente le code nécessaire à cette fonctionnalité.

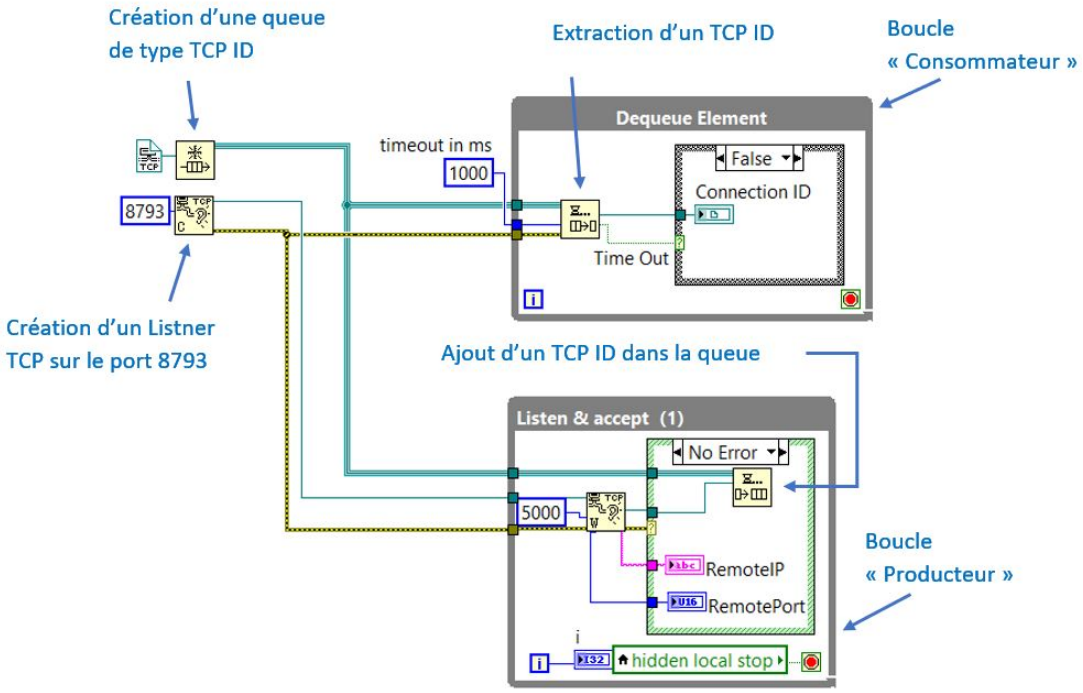


FIGURE 6.2 – Création d'une queue TCP Listner ID

Lorsqu'une requête d'ouverture de communication arrive sur le port TCP 8793, la référence à la connexion (TCP ID) est ajoutée à la queue. La boucle consommatrice interroge la queue si des éléments y sont présent. Si c'est le cas, elle extrait la référence TCP. Dans le cas contraire, la boucle attend une seconde avant la prochaine interrogation et cela grâce au "timeout".

L'évènement nécessaire au démarrage du VI dynamique est ainsi créé. Il s'agit en fait de l'extraction d'un élément de la queue. Si le "timeout" est atteint, cela signifie qu'aucune connexion est en attente dans la queue. Au contraire, si un élément est extrait, en plaçant le code de la figure 6.1 dans la structure conditionnelle de la boucle consommatrice, un clone est démarré à chaque extraction. L'ID de connexion est donné en paramètre à l'exécution du VI. Chaque clone possède ainsi son propre "TCP ID" et de cette manière, plusieurs communications peuvent être établies en parallèle.

La figure 6.3 résume cette procédure d'ouverture de communication avec le client.

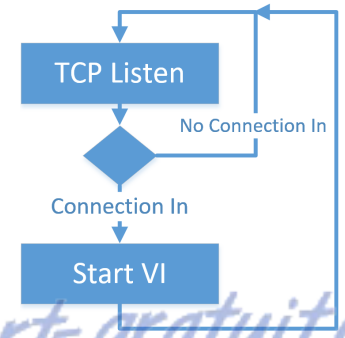


FIGURE 6.3 – Séquence d'établissement de communication

Il reste encore à déterminer si la communication est celle de la vidéo ou celle des "data" (figure 5.11). Pour cela, avant de démarrer l'exécution dynamique, l'url de la connexion entrante est analysée. Si l'url contient la valeur "ws1", un VI dédié au transfert des commandes et de mesures est démarré. Si c'est la valeur "ws2" qui est détectée, un autre VI pour la vidéo est exécuté. Si aucune de ces deux valeurs n'est détectée, la connexion est tout simplement fermée.

Il faut également détecter sur quel type de maquette le client souhaite se connecter. Pour l'instant, dans le cadre de ce travail de diplôme, deux maquettes sont utilisées (Régulation instable à bille et Entraînement électrique). Pour ce faire, une information est ajoutée à l'url de la page web. Pour la bille, le caractère "b" est ajoutée avant "ws1" et pour l'entraînement électrique, les caractères "EE" sont ajoutés. Dans les deux les cas, le même VI est exécuté. Cependant cette information permet au VI de communiquer par la suite avec la bonne maquette. Ce point est développé dans le chapitre suivant.

6.2 Gestion des connexions des Smart-Devices

Comme vu au chapitre 5.2.4, le protocole TCP est utilisé pour transférer les données entre le Web Serveur et le cRIO.

Pour ne pas transférer inutilement des données entre le serveur et l'automate, la connexion ne doit être établie que lorsqu'un client souhaite se connecter au Smart-Device. Lorsqu'il se déconnecte du laboratoire, la communication doit non seulement être fermée entre le client et le serveur mais également entre ce dernier et le Smart-Device. Pour ce faire, la même structure de lancement dynamique de VI que pour la connexion avec la page Web est utilisé. Ainsi, la communication avec l'automate n'est établie que lorsque que le VI du client est exécuté. Cependant, il faut gérer le fait que si un client est déjà connecté au même laboratoire, il ne faut pas relancer une nouvelle communication. Par conséquent, le nombre de clients connectés sur une maquette doit être connu en tout temps. De plus, il faut déterminer sur quelle maquette doit se connecter le client en fonction du type (Bille ou Entraînement électrique) et du nombre de clients connectés. Si une maquette est libre, il doit être mis en lien avec cette dernière et ne pas être mis inutilement dans une file d'attente. Le client est ainsi mis en lien avec la maquette ayant le moins de monde dans sa file d'attente. Il faut donc interroger les différents Smart-Devices de l'infrastructure pour déterminer ces informations.

Afin de répondre à ces différentes fonctionnalités, la notion de "programmation orientée objet" est introduite. Cela consiste à définir une classe pouvant contenir des fonctions et des données de différents types. Il est ensuite possible de créer des objets de cette classe et ainsi d'éviter de dupliquer les fonctions et les données dans le code.

Une classe "maquette" est ainsi définie avec les paramètres suivants :

- MaquetteId : Numéro de référence interne au cRIO de la maquette
- cRIODNS : Adresse DNS du cRIO
- ClientsListData : Cluster contenant le nombre de clients connectés, les références des VIs de la communication WebSocket du client
- sharedListClient : Référence à la variable partagée contenant le nombre de client et la liste des connexion TCP de la maquette
- MaquetteType : Type de maquette (Bille ou Entraînement électrique)
- cmdPath : Chemin de la variable partagée contenant les paramètres de contrôle de la maquette
- measPath : Chemin de la variable partagée contenant les mesures de la maquette
- videoPath : Chemin de la variable partagée contenant l'image de la maquette

Les variables partagées (sharedListClient, cmdPath, measPath, videoPath) permettent de transférer des données entre les différents clones des VIs. L'accès à ces variables se fait de manière programmatique en spécifiant le chemin de la librairie dans laquelle elles sont stockées, suivi du nom de la variable. La figure 6.4 montre un exemple de programme nécessaire à la réalisation de cette tâche.

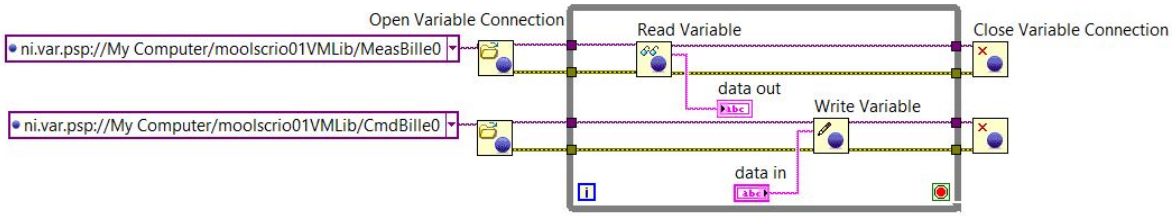


FIGURE 6.4 – Accès programmatique à une variable partagée

Dans cet exemple, la variable "MeasBill0" est lue et la variable "CmdBill0" est écrite. Les différentes variables d'une maquette sont stockées dans une librairie qui porte une partie du nom DNS de l'automate auquel elles sont liées. Par exemple, pour la libraire liée à l'automate portant le nom DNS "moolscrio01.hevs.ch", le nom de la libraire est "**moolscrio01**VMLib". De cette manière, l'accès aux variables liées à une maquette ce fait grâce au paramètre cRIODNS de l'objet de la classe.

La classe contient également les fonctions suivantes :

- CreateObj : création un objet de la classe
- EnQClient : ajout d'un client dans la liste
- DQClient : suppression d'un client de la liste
- getObjValues : retourne les paramètres de l'objet

La fonction "CreateObj" crée un nouvel objet de la classe "maquette". Cette fonction est appelée une seule fois au démarrage de l'application du Web Serveur. Elle initialise les paramètres de l'objet en prenant comme argument l'adresse DNS de l'automate, l'ID de la maquette, et le type de maquette. Le nom DNS permet d'une part d'établir la connexion TCP avec l'automate, et d'autre part de définir le chemin des variables partagées vues précédemment.

La fonction "EnQClient" est appelée lors de l'ouverture d'une connexion client (chapitre 6.1). Elle prend comme argument l'ID de la connexion TCP ainsi que la référence du VI de communication WebSocket du client. Cela permet par la suite de savoir si la communication avec le client est active ou non.

La fonction "getObjValues" fournit au VI du client le chemin des variables contenant la vidéo, les mesures, ainsi que les commandes des maquettes. Elle donne également le chemin de la variable "sharedListClient" qui permet au client de connaître sa place dans la file d'attente. C'est par ce biais-là que le VI du client sait si son rôle est observateur ou contrôleur. Si il est le premier de la file d'attente, il occupe le poste de contrôleur. Dans le cas contraire, il est observateur.

Finalement, la fonction "DQClient" est appelée toutes les secondes. Une boucle "for" appelle cette fonction pour chaque objet de classe maquette créé. Grâce au paramètre "sharedListClient" qui contient la liste des références des VIs Websocket client, la fonction peut déterminer si le VI du client est actif ou non. S'il ne l'est pas, la fonction retire le client en question de la liste.

Il reste maintenant à définir quand la communication TCP avec l'automate doit être établie. Avec la création de ces objets, la condition est simple. Lors de l'ouverture du VI du client, tous les objets sont parcourus afin de déterminer à quel automate ils appartiennent (grâce à l'adresse DNS) et afin de connaître le nombre de client connectés. Si ce nombre est égal à zéro, cela signifie qu'aucune communication n'est active et qu'il faut en démarrer une avec l'adresse DNS de l'objet.
C'est également à ce moment-là que le serveur décide sur quelle maquette il va connecter le client. En parcourant ces objets, il détermine quels sont les maquettes qui correspondent au type demandé par le client (Bille ou Entraînement électrique) et laquelle a le plus faible nombre de clients.
S'il s'avère qu'une communication doit être établie, un lancement dynamique de VI est effectué tout comme pour la communication client. Au lancement, ce VI prend comme valeur le nom DNS de l'automate, le chemin de la variable "sharedListClient" et le chemin des différentes variables de l'expérimentation (mesures, commandes, vidéo). Ce VI est arrêté lorsque le nombre de clients de l'objet vaut zéro. Pour cela, le VI interroge périodiquement la variable "sharedListClient" pour connaître ce nombre.

Le diagramme UML suivant décrit le déroulement de la connexion d'un client sur un Smart-Device.

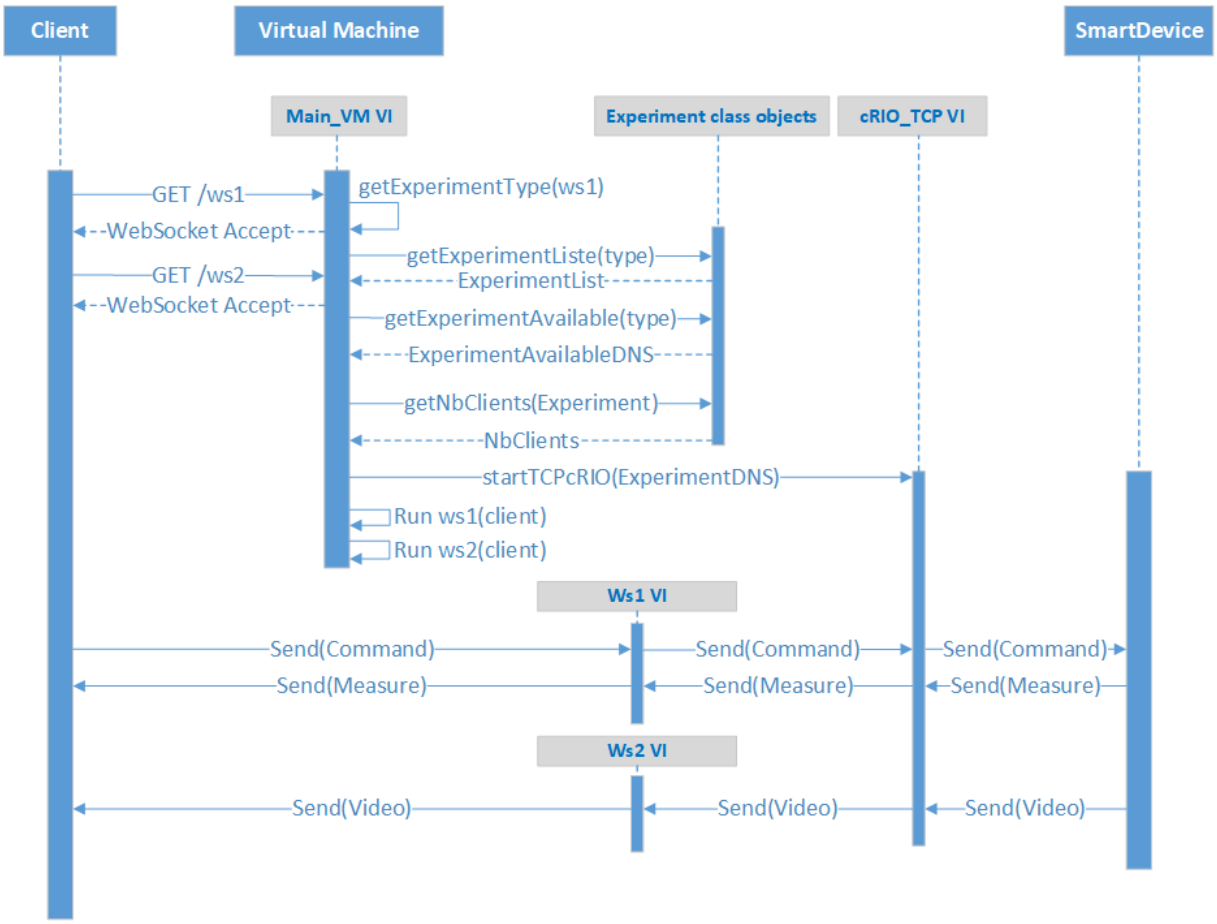


FIGURE 6.5 – Séquence UML à l'ouverture d'une connexion

6.3 Installation du programme LabVIEW sur la machine virtuelle

La machine virtuelle ne possédant pas la suite de logiciels LabVIEW, un exécutable doit être généré à partir du programme. De plus, pour qu'il puisse être exécuté, un "runtime" pour cet exécutable doit également être créé et installé sur la machine virtuelle. Cet exécutable doit contenir le VI principal ainsi que tous les sous-VIs de ce dernier. La procédure pour générer ces deux applications est disponible dans les annexes E et F.

L'exécutable LabVIEW généré nécessite qu'un compte soit ouvert en permanence sur la machine virtuelle, chose qui n'est pas concevable. Pour remédier à ce problème, la solution est de convertir l'exécutable en service Windows. Pour ce faire, le programme "NSSM" est utilisé (`https://nssm.cc/`). Celui-ci permet simplement de convertir un exécutable en service à l'aide de la commande DOS suivante : "nssm install <servicename> <programname>". Pour exécuter cette ligne de commande, il faut préalablement se placer à l'emplacement du programme "nssm.exe" avec l'invite de commandes. Une fois le service installé, aucun compte n'a besoin d'être ouvert pour exécuter le service.

7 Smart-Devices

7.1 Maquette de régulation du moteur DC

Comme vu au chapitre 3.2, les "Smart Devices" sont constitués, entre-autres, d'un entraînement électrique composé d'un moteur DC Maxon. Au moteur sont accouplés un capteur de vitesse tachymétrique, un codeur incrémental et un réducteur de rapport 60 sur lequel est également accouplé un capteur de position potentiométrique. Cet entraînement électrique est représenté sur la figure 7.1.

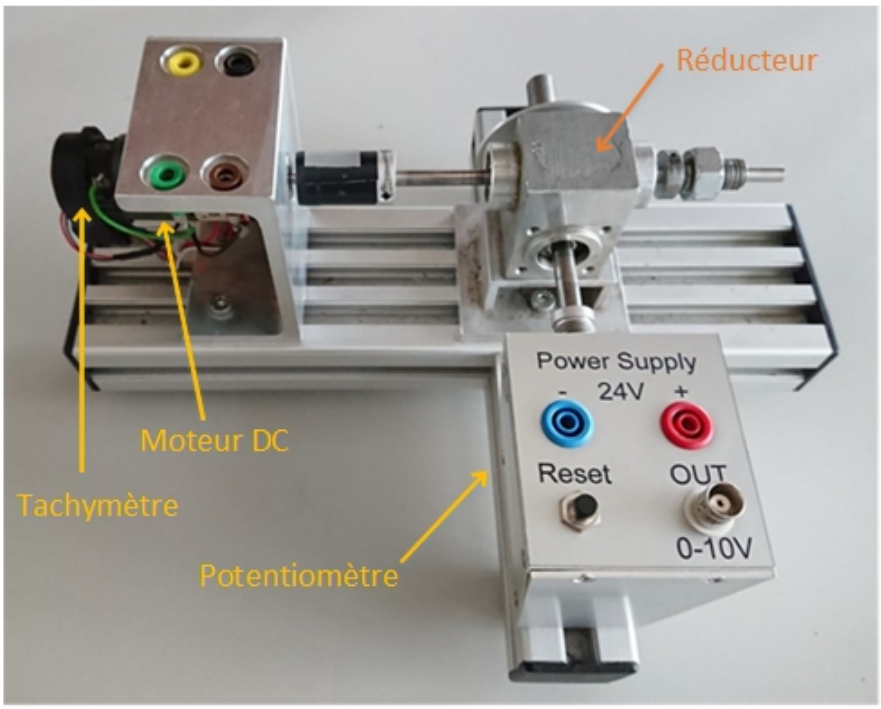


FIGURE 7.1 – Entraînement électrique

Le moteur Maxon, de type 40.032.000-32.04-005, possède les caractéristiques principales suivantes (données fabricant) :

- Puissance nominale $P_n = 15\ W$
- Vitesse max. à vide $\Omega_n = 5680\ min^{-1}$
- Tension nominale $U_n = 24\ V$
- Résistance rotorique $R_a = 7.89\ \Omega$
- Inductance rotorique $L_a = 1.54\ mH$
- Constante de temps électrique $T_a = L_a/R_a = 0.195\ ms$
- Inertie du moteur $J_m = 2.65 \cdot 10^{-6}\ kgm^2$
- Inertie de la bague $J_b = 2.2 \cdot 10^{-6}\ kgm^2$
- Inertie totale $J_{tot} = J_m + J_{ch} = 4.85 \cdot 10 - 6\ kgm^2$
- Constante K $K = 0.0394\ NmA^{-1}$

Dans un premier temps, pour le dimensionnement des régulateurs, un simple régulateur PID avec Feed-Forward est implémenté dans l'automate. Le PID utilisé est celui intégré à la librairie "Control & Simulation".

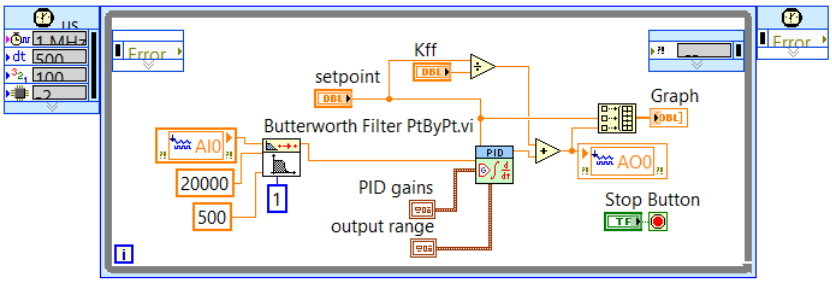


FIGURE 7.2 – Programme LabVIEW utilisé pour le dimensionnement du régulateur

Dans le cas de la régulation avec tachymètre, un filtre de Butterworth est ajouté sur la mesure. Celui-ci provient de la librairie "Signal Processing". A l'exception de ce filtre, le code est le même pour tous les dimensionnements des régulateurs.

7.2 Régulation en vitesse (tachymètre) du moteur DC

7.2.1 Dimensionnement du régulateur

Dans le but de réguler le moteur en vitesse et en position, il est nécessaire de le modéliser.

La fonction de transfert d'un moteur DC de la vitesse en fonction de la tension est la suivante :

$$F_1(s) = \frac{\omega(s)}{U(s)} = \frac{K}{LJs^2 + JRs + K^2} = \frac{0.0394}{7.469 \cdot 10^{-9} \cdot s^2 + 3.827 \cdot 10^{-5} \cdot s + 0.001552} \tag{7.1}$$

Pour le moteur utilisé, le terme "LJ" étant très petit par rapport au terme "RJ", la fonction de transfert du moteur peut être approximée par celle d'un PT1.

$$F_2(s) = \frac{\omega(s)}{U(s)} = \frac{K}{JRs + K^2} = \frac{K^{-1}}{\frac{JR}{K^2} + 1} = \frac{K_1}{Ts + 1} \tag{7.2}$$

Pour ne pas se baser sur les caractéristiques du moteur données par le fabriquant, qui peuvent varier d'un moteur à l'autre, le moteur est caractérisé par expérimentation. Pour ce faire, un saut indiciel est injecté sur l'amplificateur du moteur et la réponse est mesurée à l'aide du capteur de position et de vitesse. Ce modèle comprend donc non seulement le moteur mais également l'amplificateur et le capteur.

La régulation de vitesse est effectuée à l'aide d'un capteur tachymétrique dont la fonction de transfert peut être approximée par un simple gain. L'amplificateur, qui rentre également dans le modèle, peut aussi être caractérisé par un gain.

La mesure de la vitesse sur le tachymètre, suite à un saut indiciel de 10V à l'entrée de l'amplificateur, est représentée sur la figure 7.3.

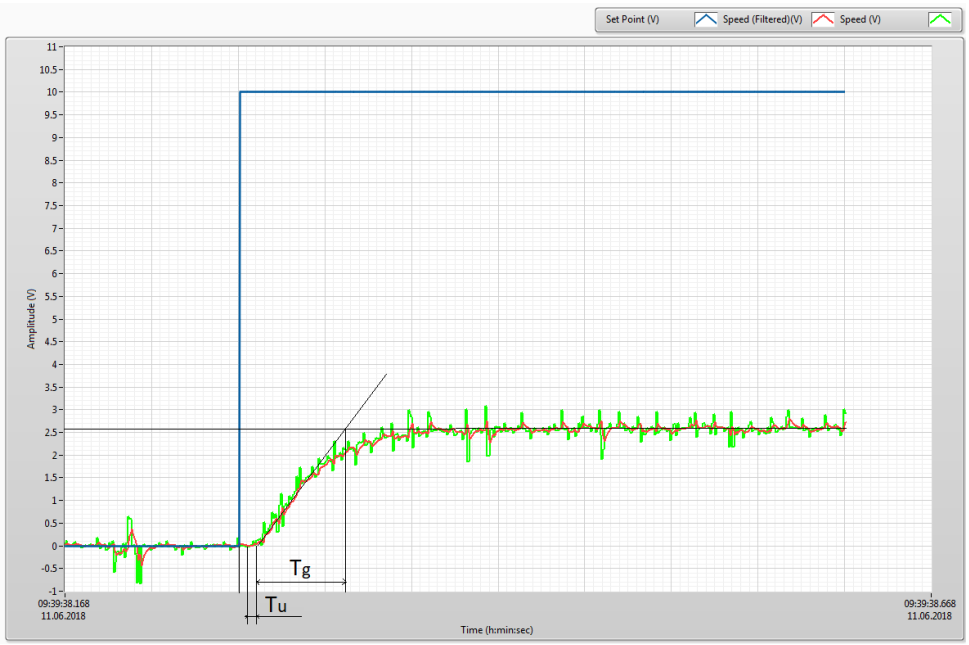


FIGURE 7.3 – Réponse indicielle amplificateur + moteur + tachymètre
(signal vert : mesure non filtrée, signal rouge : mesure filtrée, signal bleu : consigne)

Les valeurs de T_u et T_g sont relevées en traçant la tangente au point d'inflexion de la mesure du système.

Comme on peut le voir sur la figure 7.3, le signal du tachymètre est fortement bruité. Il est donc très difficile d'en tirer le gain et la constante de temps du système. La mesure est par conséquent filtrée avec un filtre du premier ordre de Butterworth. Sa fréquence de coupure est fixée par expérimentation à 500Hz. Cette fréquence permet de diminuer fortement le bruit du signal en limitant le retard engendré par le filtre (env.1ms). La fonction de transfert de ce filtre est la suivante :

$$F_{fil}(s) = \frac{1}{1 + \frac{1}{2\pi f_c}s} = \frac{1}{1 + sT} = \frac{1}{1 + 318 \cdot 10^{-6}s} \tag{7.3}$$

On remarque également sur la figure 7.3 que le signal de vitesse est retardé de 5ms par rapport au signal de commande. En mesurant la réponse de l'amplificateur, on s'aperçoit qu'il possède un retard de 5ms. Ce retard ne sera pas pris en compte pour le dimensionnement du régulateur.

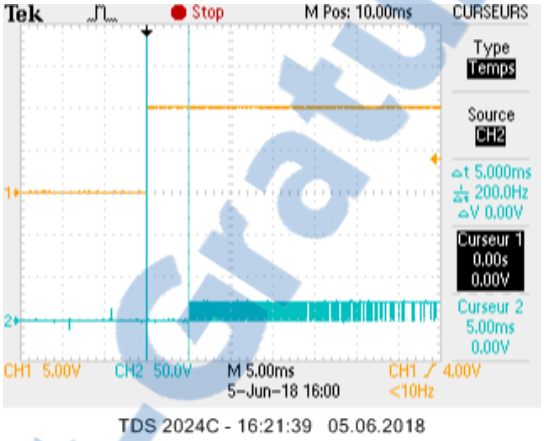


FIGURE 7.4 – Retard amplificateur PWM

De plus, même en faisant abstraction du retard de l'amplificateur, l'ordre du système est visiblement supérieur à un 1. Ceci est certainement dû à l'amplificateur et au capteur qui augmentent l'ordre du système. Par conséquent, la méthode d'approximation par un élément PTn est utilisée [2]. Cette méthode permet d'approximer l'ordre ainsi que la constante de temps d'un système suite à un saut indiciel. Cette approximation est caractérisée par des constantes de temps identiques ($T_1 = T_2 = ... = T$) et un coefficient de proportionnalité "K".

$$F(s) = \frac{K}{(T \cdot s + 1)^n} \tag{7.4}$$

2. Script SI-Mct1, v3 Fr, , mof + mad page 36

La table suivante permet ensuite de déterminer l'ordre est la constante de temps du système.

TABLE 2 – Tableau rapport Tg/Tu [3]

n	$\frac{T_u}{T}$	$\frac{T_g}{T}$	$\frac{T_g}{T_u}$
1	0	1	∞
2	0.282	2.718	9.65
3	0.805	3.695	4.59
4	1.425	4.463	3.13
5	2.100	5.119	2.44
6	2.811	5.699	2.03
7	3.549	6.226	1.75
8	4.307	6.711	1.56
9	5.081	7.164	1.41
10	5.869	7.590	1.29

Les valeurs de "T_u" et "T_g" sont ainsi relevées en soustrayant la valeur du retard à T_u. Le rapport T_g/T_u permet déterminer l'ordre du système ainsi que sa constante de temps.

$$\frac{T_g}{T_u} = 10 \rightarrow n = 2 \qquad \frac{T_g}{T} = 2.718 \Rightarrow T = 0.0184s$$

$$F(s) = \frac{K}{(1+sT)^2} = \frac{0.26}{(1+0.0184s)^2} \tag{7.5}$$

$$Avec \quad K = \frac{\Delta y}{\Delta w} = \frac{2.6}{10} = 0.26\ [-]$$

Afin de vérifier que ce modèle est correct, il est implémenté dans Matlab Simulink. Le même saut indiciel que celui utilisé pour les mesures est injecté à son entrée. Le résultat de la simulation est présenté dans la figure 7.5.

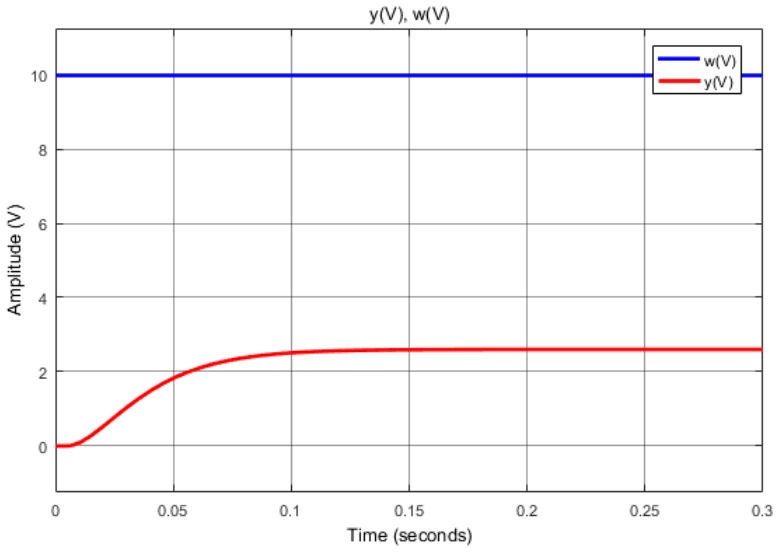


FIGURE 7.5 – Saut indiciel moteur + tachymètre + ampli (Simulink)

La simulation confirme que le modèle est correct et qu'il peut être approximé par un PT2 avec un retard de 5ms. Le gain statique et les valeurs de T_u et T_g correspondent entre la théorie et la pratique.

3. Script SI-Mct1, v3 Fr, , mof + mad page 36

Maintenant que la fonction de transfert du système à réguler est connue, il est possible de déterminer le type de régulateur ainsi que ses paramètres. Le système étant de type PT2, le régulateur à adopter sera un PI. En effet, le système n'étant pas intégrateur, il est nécessaire d'en ajouter un afin de supprimer l'erreur permanente.

La fonction de transfert du PI est la suivante :

$$G(s) = \frac{Kp(1+sTn)}{sTn} \tag{7.6}$$

La figure 7.6 représente la boucle de régulation implémentée dans Matlab Simulink. Un filtre de Butterworth d'ordre 2 et de 500Hz de fréquence de coupure est intégré à la mesure de vitesse. Cependant, il n'est pas pris en compte dans le dimensionnement du régulateur. Un retard est également placé avant le système pour simuler l'amplificateur.

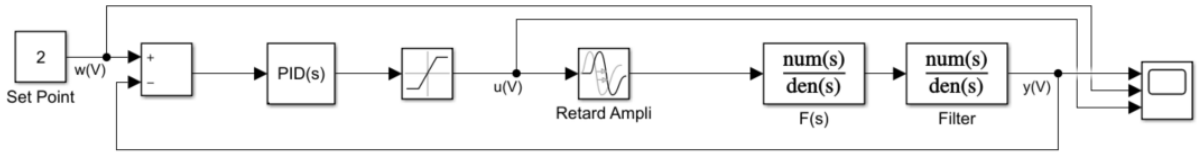


FIGURE 7.6 – Boucle de régulation en vitesse avec tachymètre

Il est ainsi possible de déterminer la fonction de transfert du système en boucle fermée.

$$Fcl(s) = \frac{KK_p(1+sT_n)}{sT_n(1+sT)^2 + KK_p(1+sT_n)} \tag{7.7}$$

Par la méthode de compensation de la constante de temps ($\mathbf{T = T_n = 0.0184s}$), la fonction de transfert peut être simplifiée comme suit :

$$Fcl(s) = \frac{KK_p}{sT_n(1+sT) + KK_p} = \frac{1}{\frac{T_nT}{KK_p}s^2 + \frac{T_n}{KK_p}s + 1} \tag{7.8}$$

Le paramètre K_p est déterminé en comparant le dénominateur de la fonction de transfert en boucle fermée avec celui d'un filtre objectif du 2ème ordre de Bessel choisi pour sa rapidité. La fonction de transfert de ce filtre est la suivante.

$$F_{fil}(s) = \frac{1}{\frac{b_1}{\omega_g^2}s^2 + \frac{a_1}{\omega_g}s + 1} \tag{7.9}$$

$$avec \quad a_1 = 1.3617 \quad et \quad b_1 = 0.618$$

Le système d'équation suivant permet ainsi de déterminer K_p et ω_g.

$$\begin{cases} \frac{T_nT}{KK_p} = \frac{b_1}{\omega_g^2} \\ \frac{T_n}{KK_p} = \frac{a_1}{\omega_g} \end{cases} \Rightarrow \begin{cases} T = \frac{b_1}{a_1\omega_g} \Rightarrow \omega_g = \frac{b_1}{a_1T} = 24.665\frac{rad}{s} \\ \mathbf{K_p} = \frac{T_n}{a_1K}\omega_g = \mathbf{1.2819}\ [-] \end{cases} \tag{7.10}$$

Il est également possible de réguler le moteur avec une simple régulateur proportionnel. Cependant, afin de compenser l'erreur permanente, il est nécessaire d'ajouter un Feed-Forward à la boucle de régulation. Le schéma de cette régulation alternative est le donné à la figure 7.7.

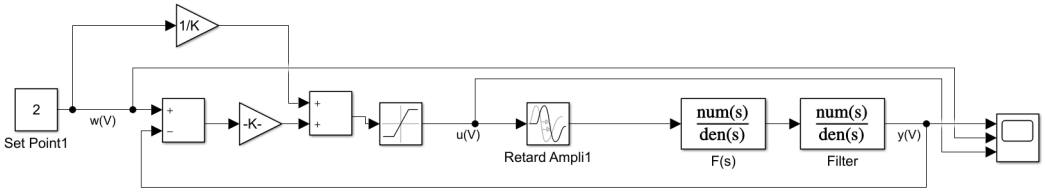


FIGURE 7.7 – Boucle de régulation en vitesse avec tachymètre et F-F

La valeur du gain du Feed-Forward vaut l'inverse du gain statique du système calculé précédemment.

$$\mathbf{K_{FF}} = \frac{1}{K} = \mathbf{3.8462}\ [-]$$

La fonction de transfert du système en boucle fermée devient :

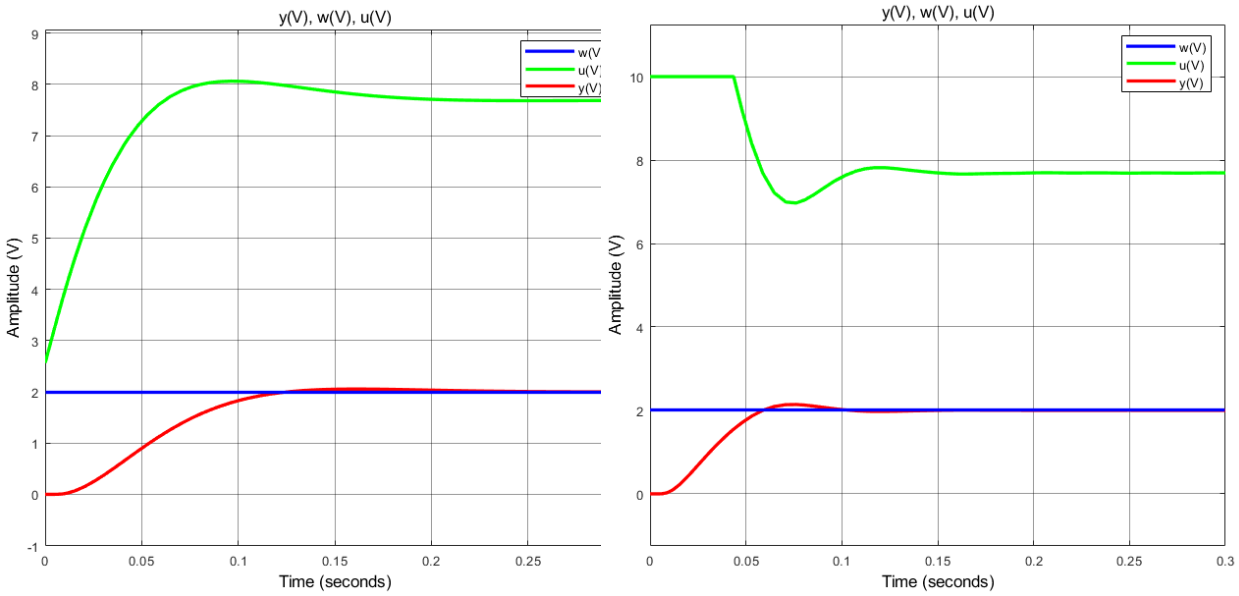
$$Fcl(s) = \frac{KK_p}{(1+sT)^2 + KK_p} = \frac{1}{\frac{T^2}{KK_p}s^2 + \frac{2T}{KK_p}s + 1} \tag{7.11}$$

Par la même méthode de comparaison par filtre objectif que sans Feed-Forward, la valeur du K_p est déterminée par le système d'équation suivant :

$$\begin{cases} \frac{T^2}{KK_p} = \frac{b_1}{\omega_g^2} \\ \frac{2T}{KK_p} = \frac{a_1}{\omega_g} \end{cases} \Rightarrow \begin{cases} \omega_g = \frac{2b_1}{a_1 T} = 49.3309\ \frac{rad}{s} \\ \mathbf{K_p} = \frac{2T}{a_1 K}\omega_g = \mathbf{5.1276}\ [-] \end{cases} \tag{7.12}$$

7.2.2 Simulations

La réponse du système en boucle fermée avec PI et avec P et commande Feed-Forward est ensuite relevée dans Matlab Simulink avec les paramètres du PI précédemment calculés.



(a) Régulation PI

(b) Régulation P et Feed-Forward

FIGURE 7.8 – Simulation régulation vitesse avec tachymètre (Simulink)

Avec le régulateur PI, la vitesse du moteur atteint ainsi la consigne en environ 125ms avec un léger dépassement de 2.7%. La valeur en sortie du régulateur PI, limitée à $\pm 10V$, ne sature pas et atteint une valeur maximale de 8.06V.

Avec le régulateur P et le Feed-Forward, le moteur atteint la consigne en 60ms avec un dépassement de 7%. Dans les deux cas, l'erreur permanente est nulle. La régulation avec Feed-Forward est donc environ deux fois plus rapide que la régulation avec PI. Ceci est dû au calcul de la fréquence du filtre objectif "ω_g" (équation (7.10) et (7.12)) qui est environ deux fois plus élevée pour la régulation avec Feed-Forward.

7.2.3 Mesures

La réponse du système avec le même saut de consigne que dans le chapitre précédent est relevée dans LabVIEW.

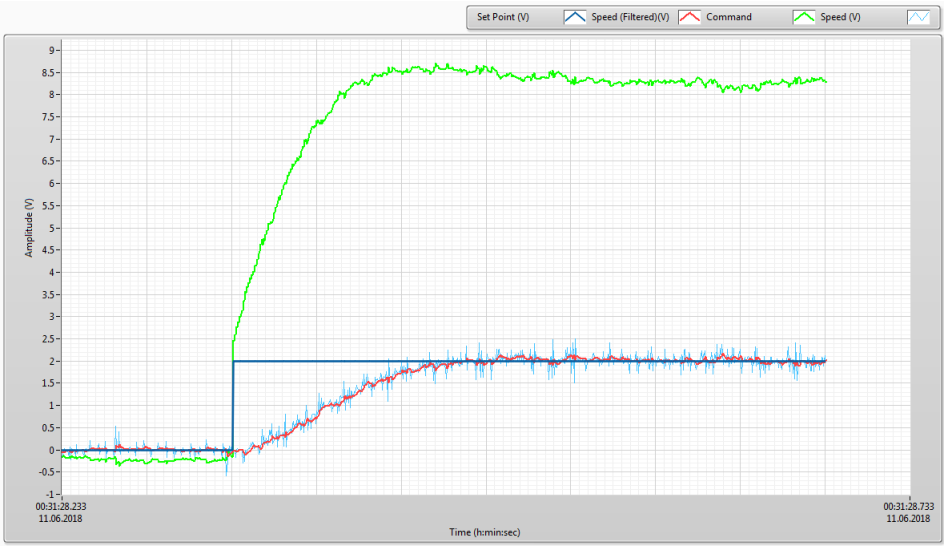


FIGURE 7.9 – Mesure régulation de vitesse PI (LabVIEW)

FIGURE 7.10 – Mesure régulation de vitesse Feed-Forward (LabVIEW)

Avec le régulateur PI, le moteur atteint la consigne en 120ms soit 5ms de moins que d'après les calculs. Ceci peut s'expliquer par le fait que le régulateur se comporte différemment entre la réalité et la simulation. On peut voir sur la figure 7.9 que le régulateur fournit une tension supérieure en réalité par rapport à la simulation, ce qui explique que la réponse soit plus rapide.

Avec le régulateur Kp et le Feed-Forward, le moteur atteint la consigne en 60ms soit le même temps qu'en simulation. Le dépassement est également le même qu'en simulation (7%).

7.2.4 Calcul de la résolution de la vitesse

En relevant le «delta» des oscillations de la mesure de vitesse (figure 7.10), sa résolution peut être déterminée. Il faut cependant tout d'abord calculer le gain du tachymètre. Pour ce faire, la vitesse du moteur est relevée en tr/s à l'aide d'un tachymètre numérique. En divisant cette valeur par l'amplitude de la tension du tachymètre du moteur à la même vitesse, le gain en tr/s/V peut être calculé.

$$K_{tachy} = \frac{n}{U} = \frac{79.5\frac{tr}{s}}{2.9V} = 27.413\frac{tr}{s \cdot V}$$
$$\Delta\Omega = \Delta U \cdot K_{tachy} = 5.48\frac{tr}{s} \tag{7.13}$$

A la vitesse nominale du moteur (5680 tr/min), cela représente une erreur relative de 5.78% sur la vitesse de rotation.

7.3 Régulation de vitesse (codeur) du moteur DC

7.3.1 Dimensionnement du régulateur

La régulation de vitesse est également effectuée à l'aide d'un codeur incrémental dont la fonction de transfert peut être approximée par un simple gain. L'amplificateur, qui rentre également dans le modèle, peut aussi être caractérisé par un gain.

La mesure de la vitesse sur le codeur dans LabVIEW, suite à un saut indiciel de 10V à l'entrée de l'amplificateur, donne la réponse suivante :

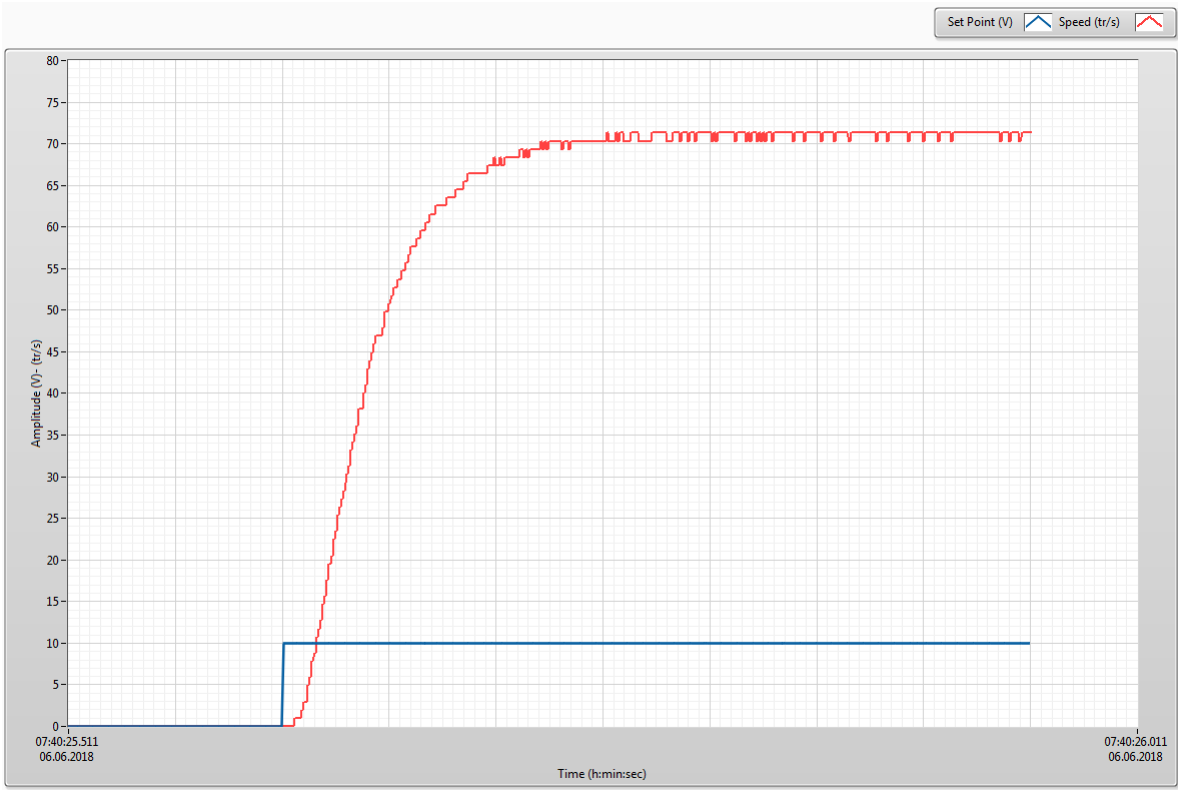


FIGURE 7.11 – Mesure réponse vitesse saut indiciel

Le gain du système amplificateur + moteur + codeur incrémental est calculé en prenant le rapport de la valeur final de la vitesse sur la différence de tension du saut indiciel.

$$K = \frac{71}{10} = 7.1\frac{tr}{s \cdot V}$$

Les valeurs de T_u et T_g valent la même chose que pour la régulation avec tachymètre. Par conséquent, l'ordre et la constante de temps du système ont également la même valeur que dans le chapitre 7.2.1. La fonction de transfert du système en boucle ouverte est donc la suivante :

$$F(s) = \frac{K}{(1+sT)^2} = \frac{7.1}{(1+0.0184s)^2} \tag{7.14}$$

Afin de vérifier que ce modèle est correct, il est implémenté dans Matlab Simulink et le même saut indiciel que pour les mesures est injecté à son entrée. Le résultat de la simulation est représenté sur la figure 7.12.

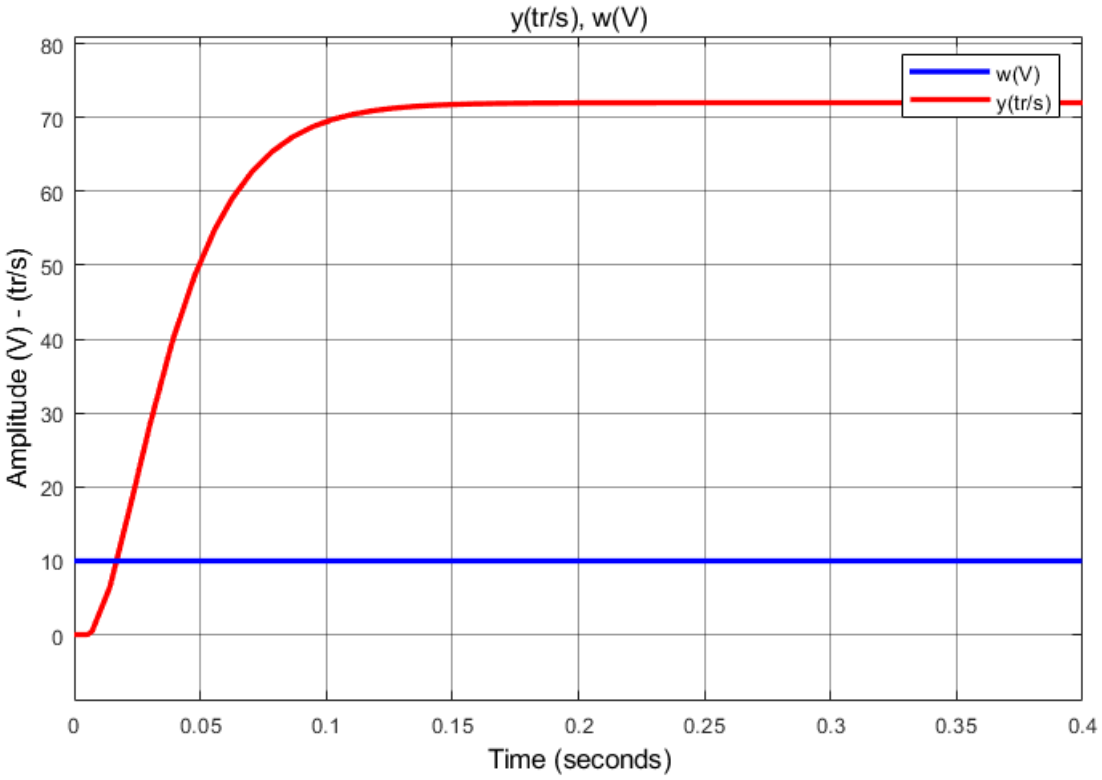


FIGURE 7.12 – Saut indiciel moteur (vitesse) + codeur + amplificateur (Simulink)

La simulation confirme que le modèle est correct et qu'il peut être approximé par un PT2 avec un retard de 5ms. Les valeurs de T_u et T_g correspondent entre les simulations et les mesures.

La fonction de transfert avec codeur étant du même type qu'avec le tachymètre (PT2), le régulateur choisi est également un PI.
La figure 7.13 représente la boucle de régulation implémentée dans Matlab Simulink.

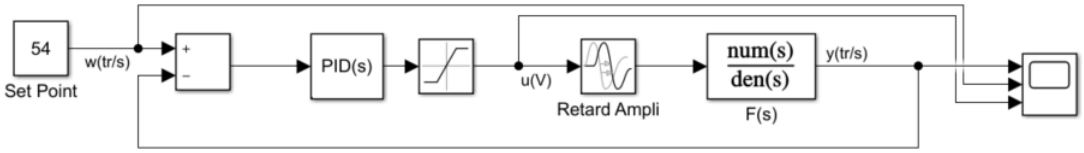


FIGURE 7.13 – Boucle de régulation en vitesse avec codeur

La fonction de transfert du système étant la même que pour le tachymètre au gain statique près, l'équation en boucle fermée est également la même. Par conséquent, les calculs de la régulation de vitesse avec tachymètre sont repris. La constante de temps est également compensée avec celle du PI ($T = \mathbf{T_n} = \mathbf{0.0184s}$)

La fréquence de coupure du filtre objectif ainsi que le gain du régulateur proportionnel sont calculés à l'aide de la relation 7.10 avec la nouvelle valeur du gain statique.

$$\begin{cases} \dfrac{T_n T}{K K_p} = \dfrac{b_1}{\omega_g^2} \\ \dfrac{T_n}{K K_p} = \dfrac{a_1}{\omega_g} \end{cases} \Rightarrow \begin{cases} T = \dfrac{b_1}{a_1 \omega_g} \Rightarrow \omega_g = \dfrac{b_1}{a_1 T} = 24.665 \ \dfrac{rad}{s} \\ \mathbf{K_p} = \dfrac{T_n}{a_1 K} \omega_g = \mathbf{0.0469} \ \dfrac{s \cdot V}{tr} \end{cases}$$

Il est également possible de réguler le moteur avec un simple régulateur proportionnel et Feed-Forward comme avec le tachymètre.

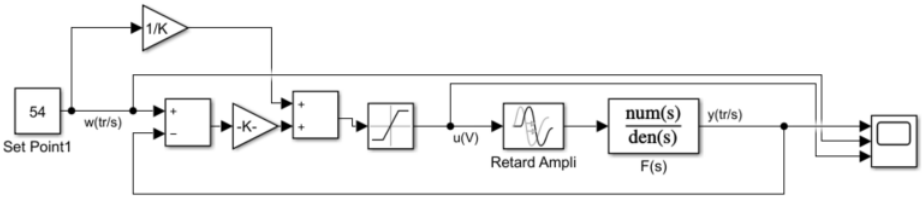


FIGURE 7.14 – Boucle de régulation en vitesse avec codeur et un régulateur P et Feed-Forward

La valeur du gain du Feed-Forward vaut l'inverse du gain statique du système calculé précédemment.

$$\mathbf{K_{FF}} = \frac{1}{K} = \mathbf{0.1389} \ \frac{s \cdot V}{tr}$$

La fonction de transfert du système en boucle fermée devient :

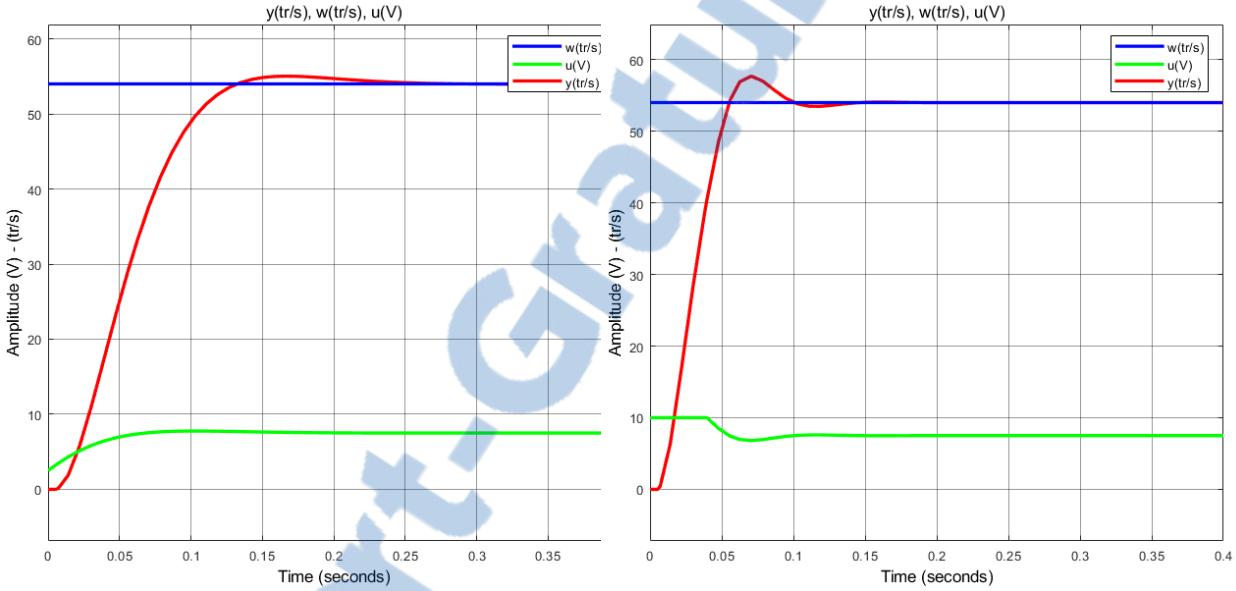
$$Fcl(s) = \frac{K K_p}{(1 + sT)^2 + K K_p} = \frac{1}{\dfrac{T^2}{K K_p} s^2 + \dfrac{2T}{K K_p} s + 1} \tag{7.15}$$

La valeur du K_p est calculée avec la même méthode de comparaison par filtre objectif que pour la régulation avec PI.

$$\begin{cases} \frac{T^2}{KK_p} = \frac{b_1}{\omega_g^2} \\ \frac{2T}{KK_p} = \frac{a_1}{\omega_g} \end{cases} \Rightarrow \begin{cases} \omega_g = \frac{2b_1}{a_1 T} = 49.3309 \frac{rad}{s} \\ K_p = \frac{2T}{a_1 K}\omega_g = 0.1852 \ \frac{s \cdot V}{tr} \end{cases} \tag{7.16}$$

7.3.2 Simulations

La réponse du système en boucle fermée est ensuite relevée dans Matlab Simulink avec les paramètres du PI et du Feed-Forward précédemment calculés.



(a) Régulation PI (b) Régulation P et Feed-Forward

FIGURE 7.15 – Simulation régulation vitesse avec codeur (Simulink)

Avec le PI, la vitesse du moteur atteint ainsi la consigne en environ 132ms avec un léger dépassement de 1.85%. La valeur en sortie du régulateur PI, limitée à $\pm 10V$, est également relevée.

La consigne est fixée à 54 tr/s, ce qui correspond à une tension de 2V sur le tachymètre. Il est ainsi possible de comparer les régulations avec tachymètre et codeur incrémental. La régulation avec codeur et plus lente de 7ms par rapport à la régulation avec tachymètre. Cette différence provient du calcul du gain statique du système qui a une influence sur le Kp.
En revanche, le dépassement est plus faible de 0.85% avec le codeur.

Avec le Feed-Forward, le moteur atteint la consigne en 55ms avec un dépassement de 6.69%. Il est donc plus rapide de 5ms par rapport à la régulation avec tachymètre. Cependant, le dépassement reste pratiquement le même entre les deux régulations (tachymètre et codeur) avec une différence de 0.31%.

7.3.3 Mesures

La réponse du système au même saut de consigne que dans le chapitre précédent est relevée dans LabVIEW.

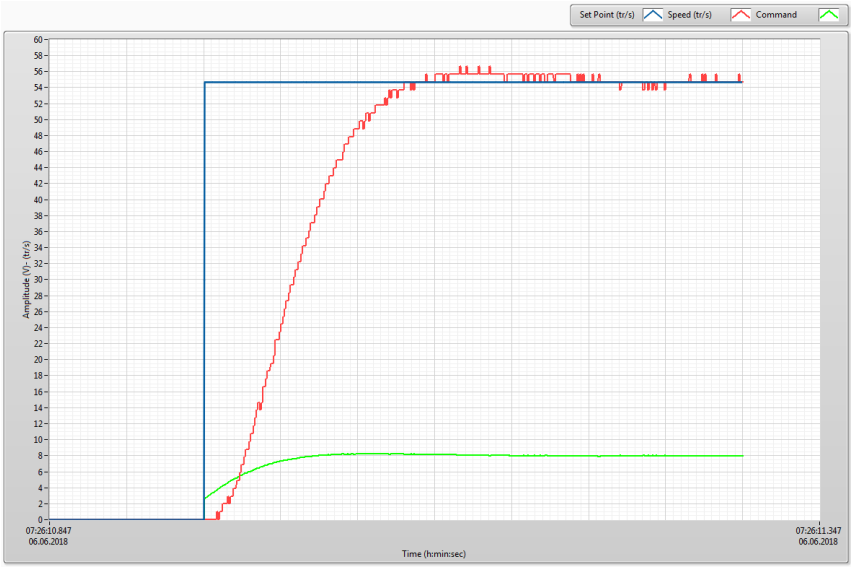


FIGURE 7.16 – Mesure régulation de vitesse avec régulateur PI (LabVIEW)

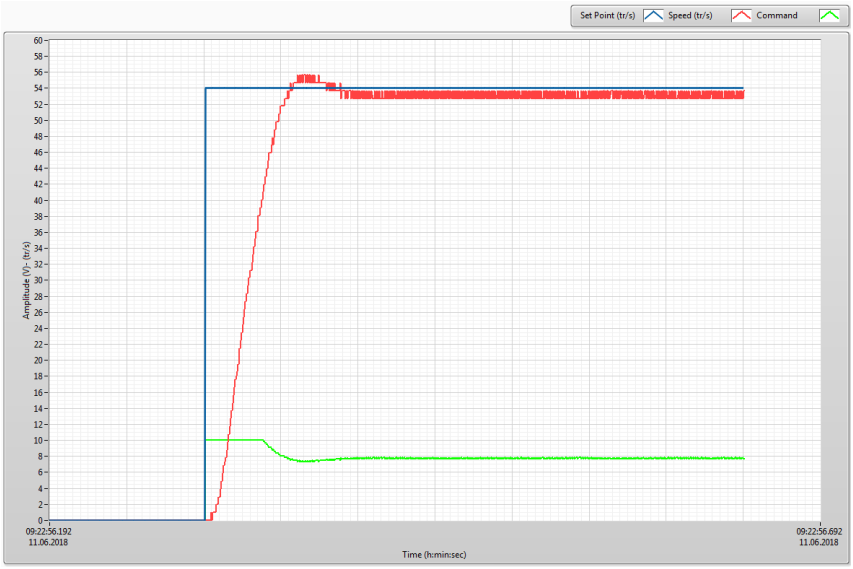


FIGURE 7.17 – Mesure régulation de vitesse avec régulateur P et Feed-Forward(LabVIEW)

Avec le régulateur PI, le moteur atteint la consigne en environ 130ms, soit le même temps que d'après les simulations. La valeur du dépassement est également de 1.85%.

Avec le Feed-Forward, le moteur atteint la consigne en environ 55ms, tout comme en simulation. Le dépassement est en revanche plus faible dans les mesures (3%) qu'en simulation (6.69%). De plus, une erreur permanente persiste. Cette erreur est due aux frottements du moteur qui agissent comme une perturbation à l'entrée du modèle du moteur qui ne peut pas être compensée par le Feed-Forward.

7.3.4 Calcul de la résolution de la vitesse

Le codeur de position possède une résolution de $0.5 \cdot 10^{-3} tr$ (2000 incréments par tour). En divisant cette valeur par la période d'échantillonnage du calcul de la vitesse ($500\mu s$), la résolution sur la vitesse peut être déterminée par l'équation 7.17.

$$\Delta\Omega = \frac{0.5 \cdot 10^{-3}}{500 \cdot 10^{-6}} = 1\frac{tr}{s} \tag{7.17}$$

A la vitesse nominale du moteur (5680 tr/min), cela représente une erreur relative de 1.05% contre 5.78% avec le tachymètre (eq. 7.13).

7.4 Régulation en position (potentiomètre) du moteur DC

7.4.1 Dimensionnement du régulateur

La régulation de position est effectuée à l'aide d'un capteur potentiométrique dont la fonction de transfert peut être approximée par un simple intégrateur avec gain. Ce capteur étant connecté après le réducteur, le gain est également dépendant du rapport de réduction du réducteur (60). L'amplificateur, qui rentre également dans le modèle, est caractérisé par un gain.

La mesure de la position sur le potentiomètre, suite à un saut indiciel de 10V à l'entrée de l'amplificateur, est représentée sur la figure 7.18.

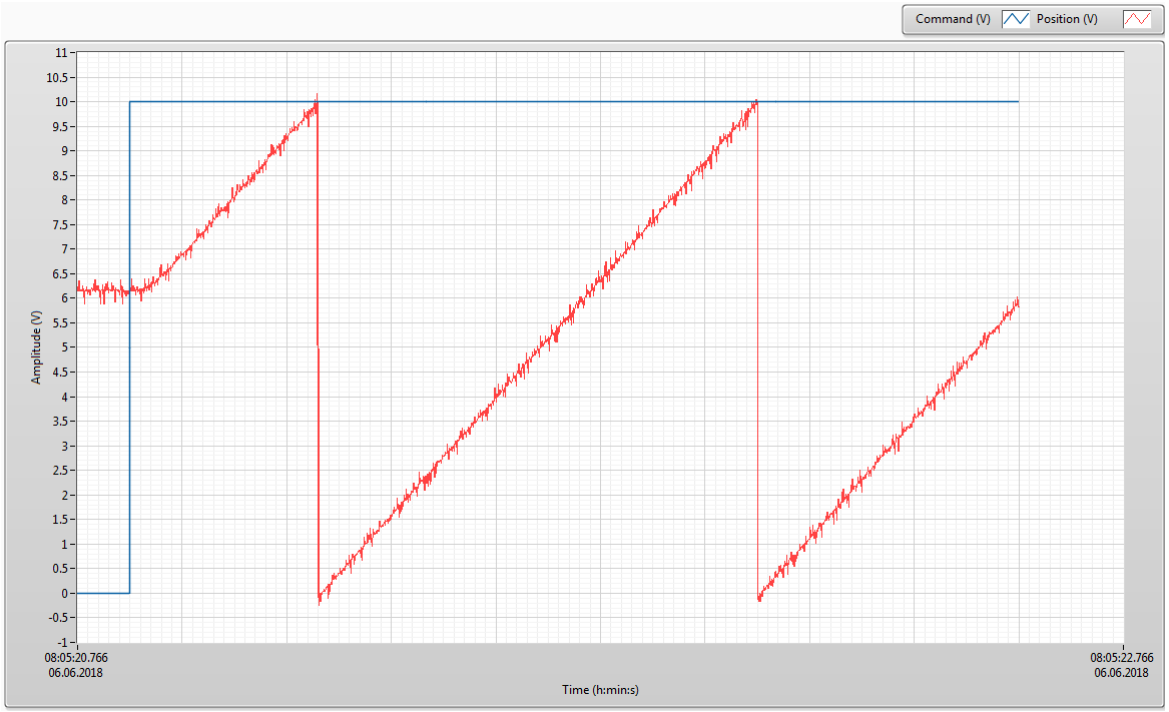


FIGURE 7.18 – Réponse indicielle position amplificateur + moteur + potentiomètre

La fonction de transfert du moteur en position peut ainsi être déterminée avec le modèle d'un PT2 avec intégrateur. En supposant la constante de temps du tachymètre infiniment petite, la constante de temps du moteur peut être tirée du chapitre 7.2.1, à savoir 0.0184s.
Le gain du système est calculé en prenant le rapport de la pente du signal de position sur le delta du saut indiciel. Ce gain vaut ainsi :

$$K = \frac{\Delta y}{\Delta t} \cdot \frac{1}{\Delta u} = \frac{10.2}{0.82} \cdot \frac{1}{10} = 1.2439\ [-]$$

Il est à noter que 10 volts correspondent à environ 60 tours sur l'axe du moteur et à 1 tour sur l'axe du réducteur.
L'équation du système en boucle ouverte vaut par conséquent :

$$F(s) = \frac{K}{s(1+sT)^2} \simeq \frac{K}{s(1+2Ts)} = \frac{1.2439}{s(1+0.0368s)} \tag{7.18}$$

Le système en boucle fermée étant d'ordre 3 avec un régulateur P, ou d'ordre 4 avec un régulateur PI, si on le compare avec un filtre objectif du même ordre, on obtient un système d'équations à x inconnues et x+1 équations. Un tel système d'équations n'a pas forcément de solution. Par conséquent, la fonction de transfert du moteur est simplifiée par un PT2 pour obtenir un système de x équations à x inconnues.

Afin de vérifier que ce modèle est correct, il est implémenté dans Matlab Simulink et le même saut indiciel que pour les mesures est injecté à son entrée. Comme pour la régulation de vitesse, un retard est ajouté sur le signal de commande du moteur. Le résultat de la simulation est le suivant.

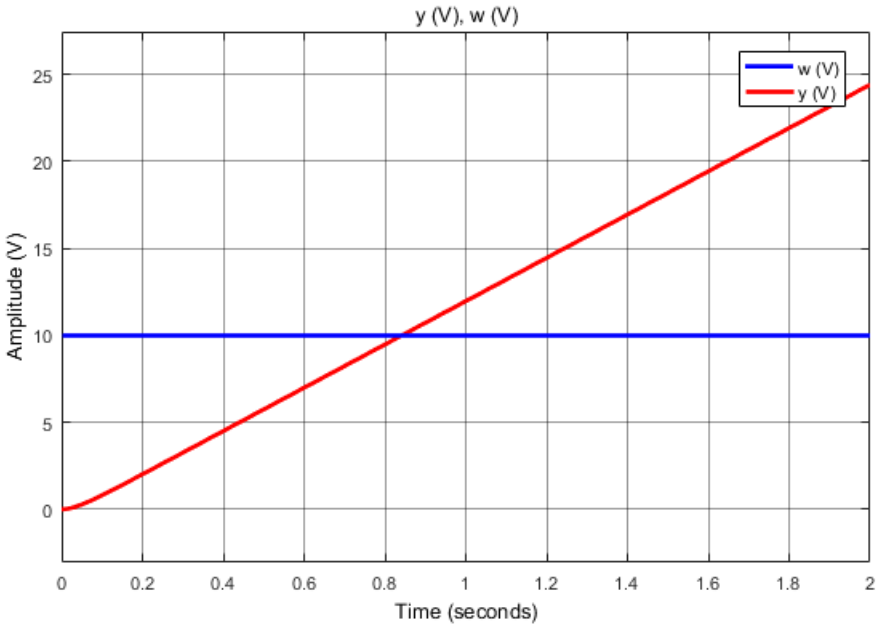


FIGURE 7.19 – Saut indiciel moteur + ampli + potentiomètre (Simulink)

La simulation confirme que le modèle est correct.

Maintenant que la fonction de transfert du système à réguler est connue, il est possible de déterminer le type de régulateur ainsi que ses paramètres. Le système étant de type PT2 avec intégrateur, le régulateur à adopter est un simple P. En effet, le système possédant un intégrateur, le système corrige de lui-même l'erreur permanente.

La figure suivante représente la boucle de régulation implémentée dans Matlab Simulink.

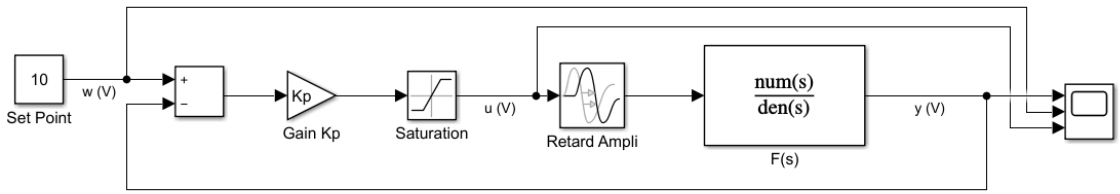


FIGURE 7.20 – Boucle de régulation en position avec potentiomètre

Il est ainsi possible de déterminer la fonction de transfert du système en boucle fermée.

$$Fcl(s) = \frac{KK_p}{s(1+2Ts)+KK_p} = \frac{1}{1+\frac{1}{KK_p}s+\frac{2T}{KK_p}s^2} \tag{7.19}$$

Par la méthode de comparaison avec filtre objectif, le gain du régulateur proportionnel peut être calculé. Le système en boucle fermée est ainsi comparé avec un filtre de Bessel du 2ème ordre dont la fonction de transfert est la suivante :

$$F_{fil}(s) = \frac{1}{1+\frac{a_2}{\omega_g}s+\frac{b_2}{\omega_g^2}s^2} \tag{7.20}$$

$$avec \quad a_1 = 1.3617 \quad b_1 = 0.618$$

En égalisant les dénominateurs des équations (7.19) et (7.20), nous obtenons le système d'équations suivant et les valeurs de ω_g et K_p peuvent être définies.

$$\begin{cases} \frac{b_1}{\omega_g^2} = \frac{2T}{KK_p} \\ \frac{a_1}{\omega_g} = \frac{1}{KK_p} \end{cases} \Rightarrow \begin{cases} \omega_g = \frac{b_1}{2a_1T} = 12.3327 \ \frac{rad}{s} \\ \mathbf{K_p} = \frac{\omega_g}{a_1K} = \mathbf{7.281} \ [-] \end{cases} \tag{7.21}$$

7.4.2 Simulations

La réponse du système est ensuite relevée dans Matlab Simulink avec les paramètres du régulateur P précédemment calculés.

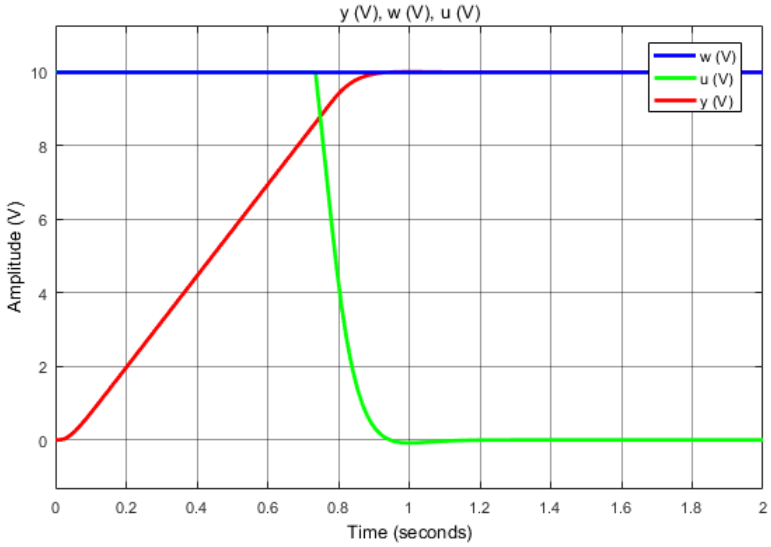


FIGURE 7.21 – Simulation régulation de position avec potentiomètre (Simulink)

Le moteur atteint ainsi la consigne de 10V en environ 0.9s sans dépassement. La valeur en sortie du régulateur P, limitée à $\pm 10V$, est également relevée.

Pour une consigne de 10V, qui correspond à 60 tours du moteur, la sortie du régulateur sature à 10V durant les 0.7 premières secondes puis redescend à zéro.

7.4.3 Mesures

La réponse du système avec le même saut de consigne que dans le chapitre précédent est relevée dans LabVIEW.

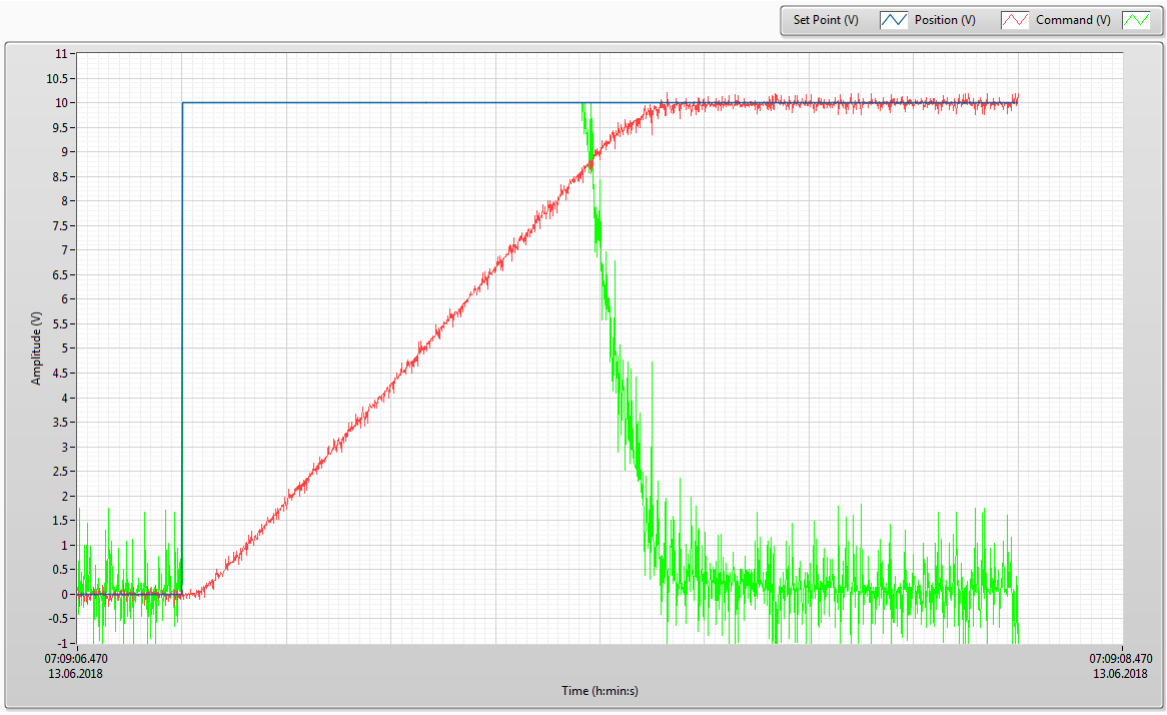


FIGURE 7.22 – Mesure régulation P de position avec potentiomètre (LabVIEW)

D'après les mesures, le moteur atteint la consigne en 1s soit le même temps qu'en simulation. Comme en simulation, l'erreur permanente et le dépassement sont nuls. L'absence de l'erreur permanente avec un régulateur de position P indique que les frottements du moteur ne sont pas très importants.

7.4.4 Calcul de la résolution

Le potentiomètre étant connecté après le réducteur, la résolution de ce dernier est multipliée d'un facteur 60. Sachant que 10 volts sur le potentiomètre correspondent à 1 tour complet sur l'axe du réducteur (1/60 sur l'axe du moteur), sa résolution, ou l'erreur absolue sur la position, peut être calculée d'après les variations de la tension mesurée au point précédent.

$$\Delta\Theta = \frac{\Delta U}{10 \cdot 1tr} \cdot 60 = \frac{0.25}{10 \cdot 1} \cdot 60 = 1.5\ tr \tag{7.22}$$

7.5 Régulation de position (codeur) du moteur DC

7.5.1 Dimensionnement du régulateur

La régulation de position est également effectuée à l'aide d'un codeur incrémental dont la fonction de transfert peut être approximée par un simple intégrateur avec gain. L'amplificateur, qui rentre également dans le modèle, est caractérisé par un gain.

La mesure de la position du codeur, suite à un saut indiciel de 10V à l'entrée de l'amplificateur, donne la réponse suivante.

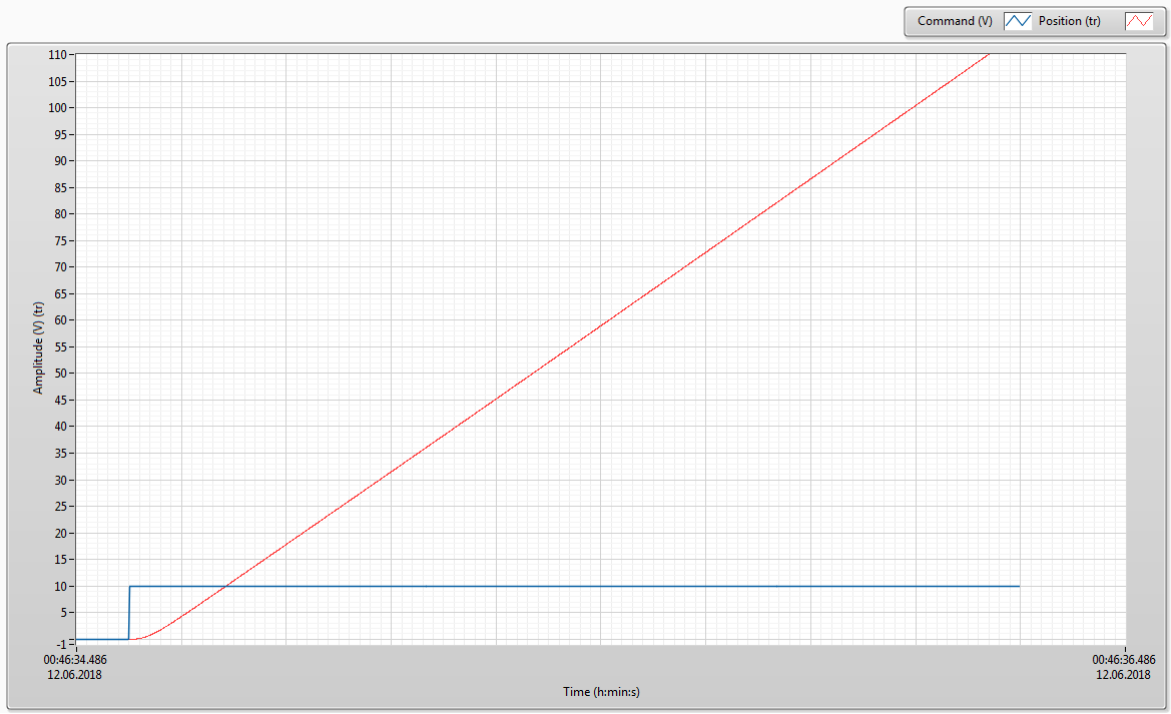


FIGURE 7.23 – Réponse indicielle position amplificateur + moteur + codeur

La fonction de transfert du moteur en position peut ainsi être déterminée avec le modèle d'un PT2 avec intégrateur. La constante de temps reste la même que pour les régulations précédentes. Le gain du système est calculé en prenant le rapport de la pente du signal de position sur le delta du saut indiciel. Ce gain vaut ainsi :

$$K = \frac{\Delta y}{\Delta t} \cdot \frac{1}{\Delta u} = \frac{90}{1.27} \cdot \frac{1}{10} = 7.0866 \ \frac{tr}{V}$$

L'équation du système en boucle ouverte est également simplifiée, comme pour le réglage de position avec potentiomètre, et ceci pour les mêmes raisons qu'évoquées dans le chapitre 7.4.1.

$$F(s) = \frac{K}{s(1+sT)^2} = \frac{K}{s(1+2Ts)} = \frac{7.0866}{s(1+0.0368s)} \tag{7.23}$$

Afin de vérifier que ce modèle est correct, il est implémenté dans Matlab Simulink et le même saut indiciel que pour les mesures est injecté à son entrée. Comme pour les régulations précédentes, un retard est ajouté à l'entrée du système pour simuler l'amplificateur. Le résultat de la simulation est représenté sur la figure 7.24.

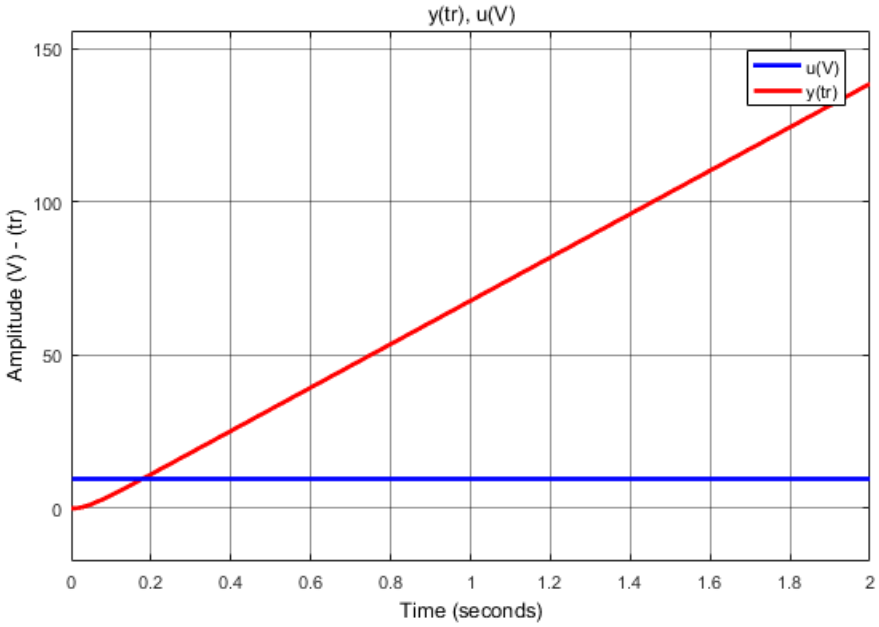


FIGURE 7.24 – Saut indiciel moteur + ampli + codeur (Simulink)

La simulation confirme que le modèle est correct.

La fonction de transfert du système étant la même que pour la régulation avec potentiomètre au gain statique près, le même type de régulateur est employé.

La figure 7.25 représente la boucle de régulation implémentée dans Matlab Simulink.

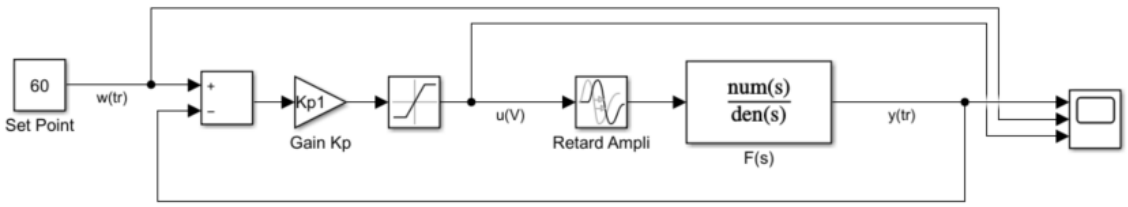


FIGURE 7.25 – Boucle de régulation en position avec codeur

La fonction de transfert en boucle fermée est ainsi de la même forme qu'avec le potentiomètre. La même méthode par comparaison avec filtre objectif est employée. Par conséquent, le système d'équation 7.21 est repris pour calculer la fréquence de coupure du filtre objectif ainsi que le gain du régulateur proportionnel avec la nouvelle valeur du gain statique du système.

$$\begin{cases} \frac{b_1}{\omega_g^2} = \frac{2T}{KK_p} \\ \frac{a_1}{\omega_g} = \frac{1}{KK_p} \end{cases} \Rightarrow \begin{cases} \omega_g = \frac{b_1}{2a_1T} = 12.3327 \ \frac{rad}{s} \\ \mathbf{K_p} = \frac{\omega_g}{a_1K} = \mathbf{1.278} \ \frac{V}{tr} \end{cases}$$

7.5.2 Simulations

La réponse du système est ensuite relevée dans Matlab Simulink avec les paramètres du régulateur P précédemment calculés.

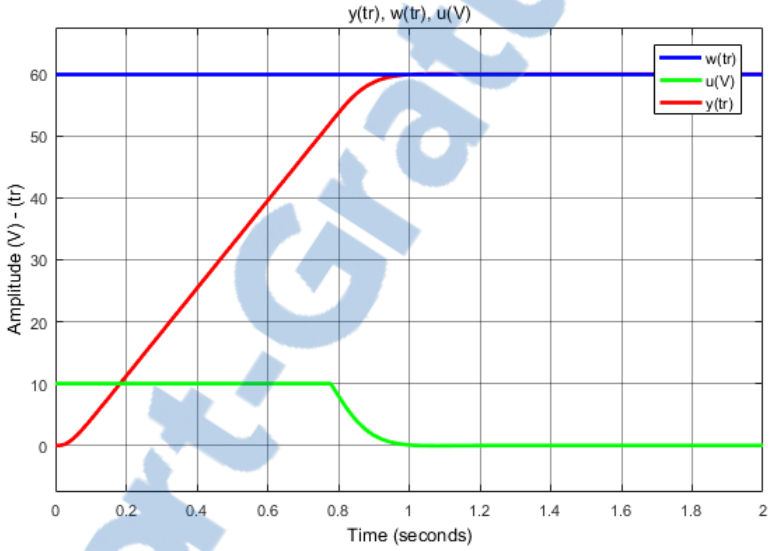


FIGURE 7.26 – Simulation régulation P de position avec codeur (Simulink)

Le moteur atteint ainsi la consigne de 60 tours en environ 0.95s avec un dépassement de 0.56%. La valeur en sortie du régulateur P, limitée à $\pm 10V$, est également relevée.

Pour une consigne de 60 tours du moteur, la sortie du régulateur sature à 10V durant les 0.75 premières secondes puis redescend à zéro.

La régulation du système avec codeur est plus rapide d'environ 50ms par rapport à la régulation avec potentiomètre. Ceci est dû à la modélisation du moteur et du gain statique du système qui n'est pas le même dans les deux cas et qui a une influence sur le calcul du gain du régulateur.

7.5.3 Mesures

La réponse du système avec le même saut de consigne que dans le chapitre précédent est relevée dans LabVIEW.

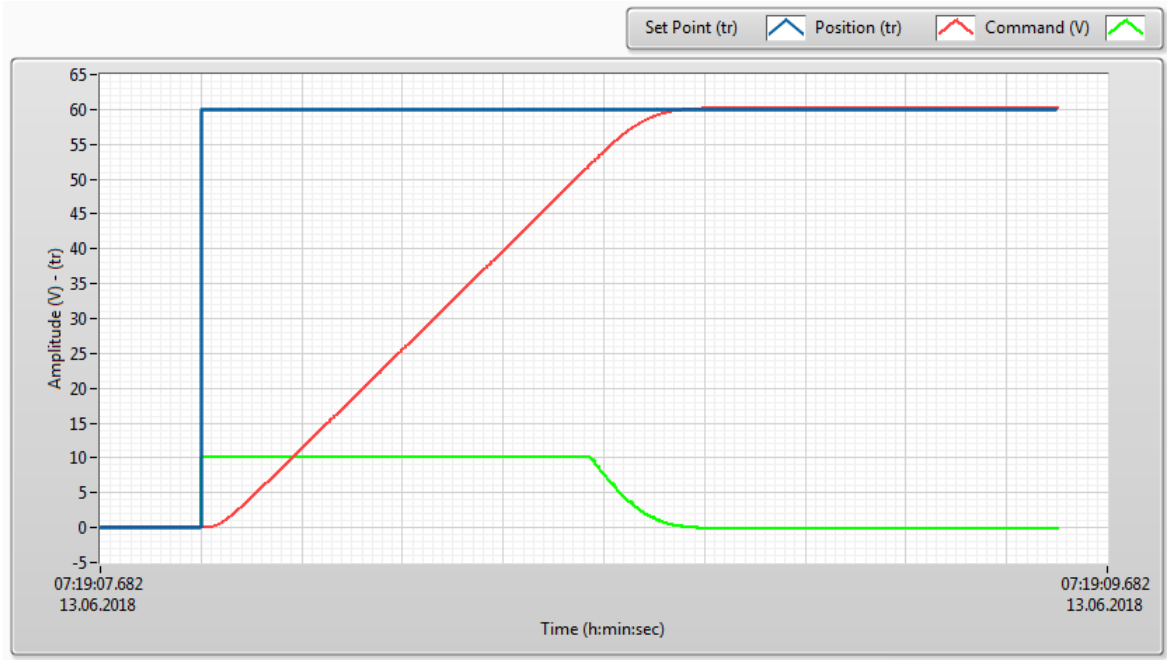


FIGURE 7.27 – Régulation P du moteur en position avec codeur (LabVIEW)

D'après les mesures, le moteur atteint la consigne en 0.95s, soit le même temps qu'en simulation.

Le dépassement vaut également environ 0.5%, comme en simulation.

7.5.4 Calcul de la résolution

Le codeur incrémental délivre 2000 incréments par tour, soit une résolution de $0.5 \cdot 10^{-3}$ tr sur l'arbre du moteur. Par comparaison, le potentiomètre a une résolution $25 \cdot 10^{-3}$ tr après le réducteur et 1.5 tr sur l'arbre du moteur. Même sans prendre en compte le réducteur, le codeur a une bien meilleure résolution que le potentiomètre (5 fois plus petite). La régulation du moteur est par conséquent plus précise avec le codeur.

7.6 Maquette de régulation de système instable à bille

La figure suivante représente la maquette de régulation instable à bille. Le but de cette expérimentation est de réguler la position d'une bille posée sur un rail à l'aide d'un capteur de position (cf. annexe B) optique et d'un servomoteur.

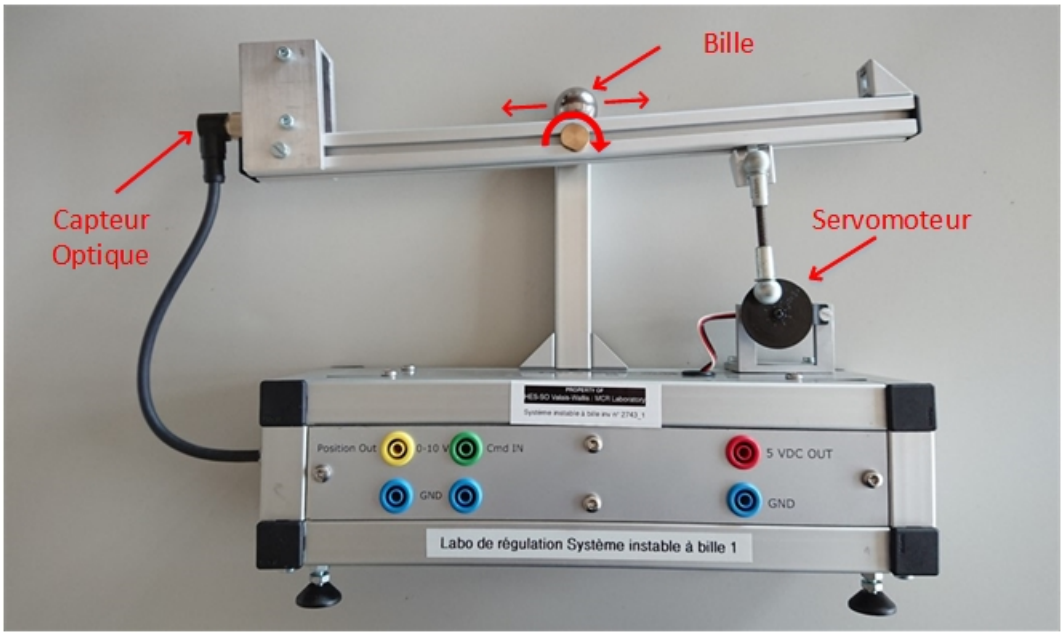


FIGURE 7.28 – Maquette de régulation de système instable à bille

7.6.1 Dimensionnement du régulateur

Un saut de tension est injecté sur le système afin de le caractériser par une fonction de transfert. La figure 7.29 représente la réponse du système suite à ce saut indiciel.

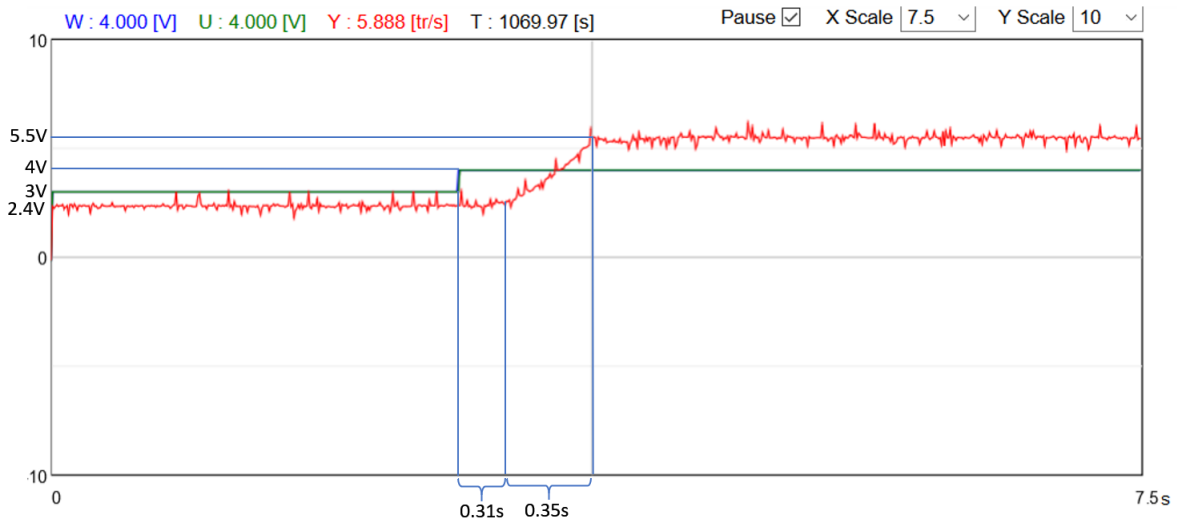


FIGURE 7.29 – Saut indiciel Système instable à bille

D'après ces mesures, le système en boucle ouverte se rapproche d'un intégrateur saturé à 5.5V avec un gain à déterminer et un retard.

Le gain du système intégrateur est déterminé comme suit :

$$K = \frac{5.5 - 2.4}{0.35 \cdot 1} = 8.857\ [\frac{V}{s}]$$

La fonction de transfert du système vaut ainsi :

$$F(s) = \frac{K}{s} \cdot e^{-sT} = \frac{8.857}{s} \cdot e^{-0.31s}$$

Afin de vérifier que ce modèle est correct, il est implémenté dans Matlab Simulink et le même saut indiciel que pour les mesures est injecté à son entrée. Le résultat de la simulation est visible sur la figure 7.30. Dans la simulation, la bille ne part pas de 2.4 V mais de 0V.

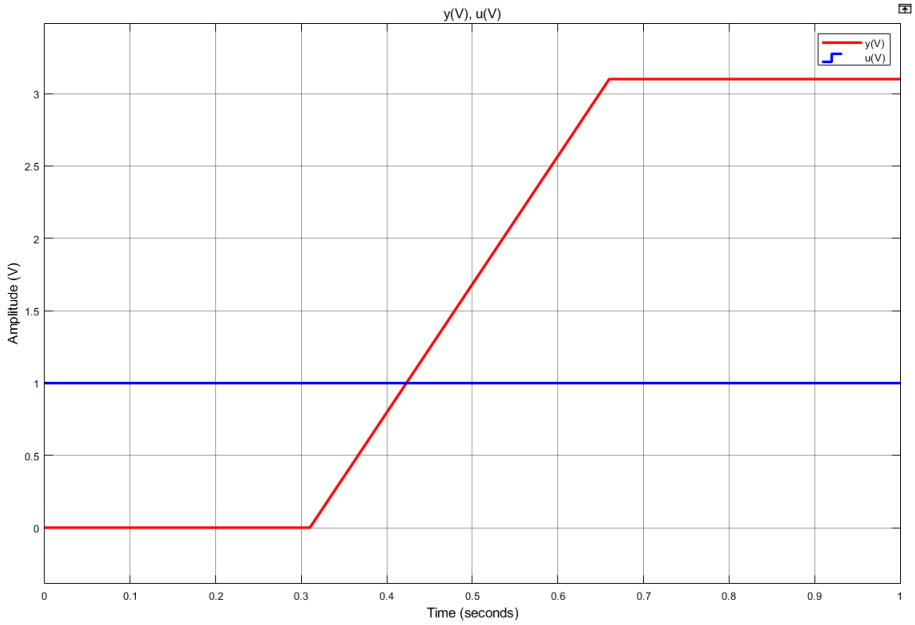


FIGURE 7.30 – Simulation du système instable à bille en boucle ouverte

La simulation correspond bien à la réalité.

Pour le dimensionnement du régulateur, le retard du système n'est pas pris en compte.

Pour vaincre les frottements de ce système intégrateur, on propose d'utiliser un régulateur PI (eq. 7.6). La fonction de transfert en boucle fermée vaut ainsi :

$$F_{cl}(s) = \frac{KK_p(1+sT_n)}{T_n s^2 + KK_p(1+sT_n)} = \frac{KK_p(1+sT_n)}{T_n s^2 + KK_p T_n s + KK_p} = \frac{1+sT_n}{\frac{T_n}{KK_p}s^2 + T_n s + 1}$$

Le dénominateur de la fonction de transfert est comparé avec celui d'un filtre objectif de Butterworth du 2ème ordre (eq. 7.9).

$$\frac{T_n}{KK_p}s^2 + T_n s + 1 = \frac{b_1}{\omega_g^2}s^2 + \frac{a_1}{\omega_g}s + 1 \tag{7.24}$$

Nous avons une seule équation (7.24) qui possède deux inconnues (K_p et ω_g). La pulsation de coupure est donc fixée empiriquement à 0.8 rad/s pour obtenir un système le plus rapide possible sans oscillations.

$$\begin{cases} \dfrac{b_1}{\omega_g^2} = \dfrac{T_n}{KK_p} \\ \dfrac{a_1}{\omega_g} = T_n = 1.7021 \end{cases} \Rightarrow K_p = \frac{\omega_g a_1}{Kb_2} = 0.123$$

7.6.2 Simulations

La simulation du système en boucle fermée avec les paramètres précédemment calculés donne les résultats suivants.

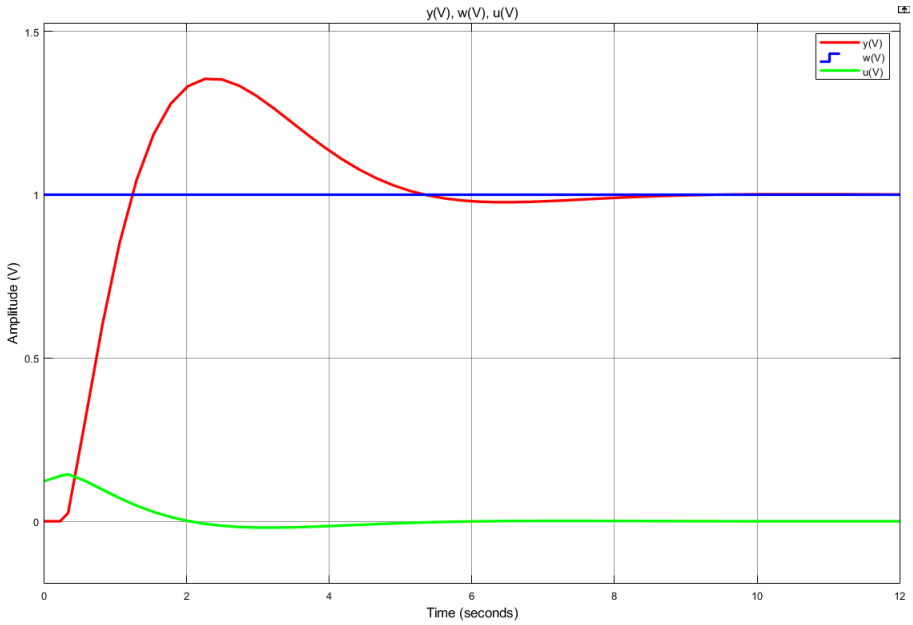


FIGURE 7.31 – Simulation du système instable à bille en boucle fermée

Le système se stabilise sur la consigne en 9 secondes sans erreur permanente avec un dépassement de 35%.

7.6.3 Mesures

Une mesure est ensuite effectuée sur la maquette avec les paramètres du PI calculés. Les résultats ne sont pas probants du tout. Le système oscille est ne se stabilise jamais sur la consigne.

Cela provient de la modélisation du système qui ne représente pas bien la réalité. Ceci est certainement dû aux frottements secs de la bille sur le rail, qui varient en fonction de l'angle de ce dernier.

La figure 7.32 permet de déterminer ces frottements en fonction de l'angle du rail.

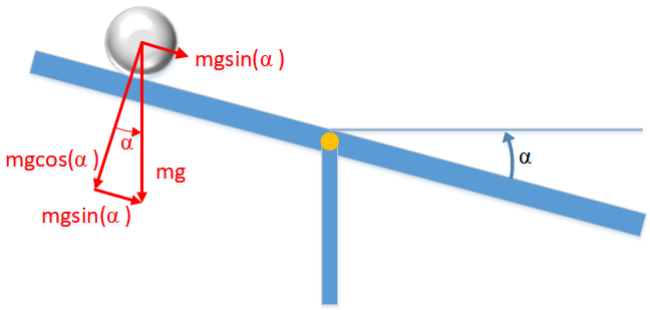


FIGURE 7.32 – Force sur la bille en fonction de l'angle du rail

Les frottements secs peuvent ainsi être déterminés avec le coefficient de frottement "μ".

$$F_f = \mu \cdot m \cdot g \cdot cos(\alpha)$$

De plus, le moteur qui a du jeu au niveau de son accouplement n'arrange pas les choses. A cela s'ajoute le fait que le capteur a une zone morte sur les 50 premiers millimètres du rail.
Par conséquent, au vu de la complexité du système, les paramètres du PI sont déterminés empiriquement.

Après plusieurs essais, la valeur de K_p est gardée proche de la valeur calculée de 0.123, soit $K_p = 0.15$ et la valeur de T_i est augmentée à 20 s pour obtenir un PI avec un élément intégrateur beaucoup plus faible étant donné que le système est fortement intégrateur.

La figure 7.33 représente la mesure du système avec ces valeurs de PI suite à un saut de consigne de 4V.

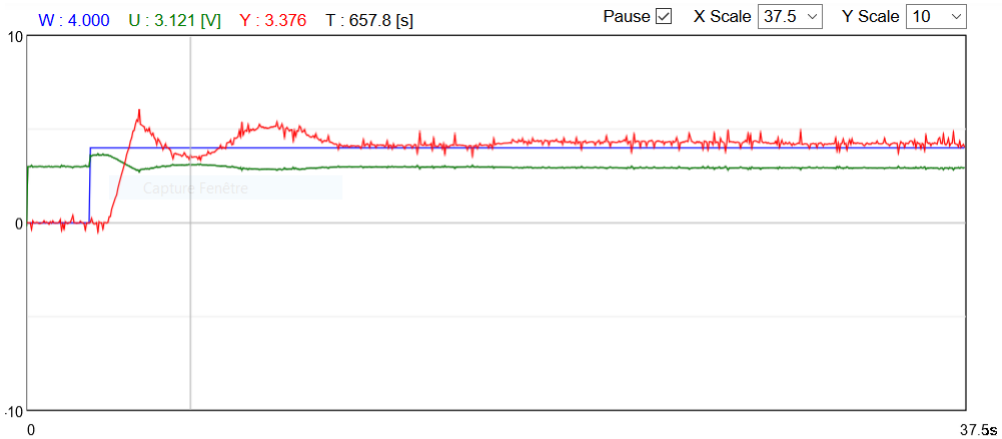


FIGURE 7.33 – Régulation de la position de la bille (mesures)

La consigne est fixée à 4V, ce qui correspond à une distance de 190 mm depuis le capteur optique. La bille atteint la consigne en 1s, avec un dépassement de 37.5 %. Sans la butée en bout de rail, ce dépassement aurait été plus élevé. La bille se stabilise ensuite sur la consigne en 9.9 s avec une erreur permanente d'environ 5%.
Etant donné que le système comporte un retard, la régulation du système instable à bille pourrait être améliorée avec un prédicteur de Smith. Ce dernier n'a pas été implémenté dans le cadre de ce travail.

8 Programmation du cRIO

Les deux maquettes régulées et décrites précédemment sont pilotées avec un automate industriel National Instrument cRIO-9067. Les signaux analogiques des différents capteurs sont mesurés à l'aide d'un module analogique de 8 entrées de tension (NI-9201). Les signaux de commande sont générés à l'aide d'un module de sortie analogique à 4 sorties de tension ±10V (NI-9263). La position et la vitesse du codeur incrémental de l'entraînement électrique sont déterminées grâce à un module d'entrées numérique (NI-9411). Les caractéristiques principales de ces produits se trouvent dans l'annexe D.

Le cRIO-9067 possède une FPGA (Field-Programmable Gate Array) et un processeur temps réel avec le système d'exploitation NI Linux Real-Time (CPU). Dans LabVIEW, l'automate peut être programmé soit en mode "Scan Interface" soit en mode "FPGA Interface". Le deuxième mode permet de programmer la FPGA en plus de la CPU contrairement au premier mode. Le code contenu dans la FPGA peut être exécuté plus rapidement que celui dans la CPU puisqu'il est directement exécuté par un circuit de portes logiques. Le temps de réponse des entrées/sorties est ainsi plus rapide. Cependant, la FPGA ne contient que des fonctions de base LabVIEW (opérateurs mathématiques, portes logiques etc.). Les fonctions plus avancées, comme la communication TCP par exemple, doivent être programmées dans la CPU de l'automate.

En programmant la partie régulation dans la FPGA et la partie communication dans la CPU, l'utilisation du processeur de l'automate peut être soulagée et ainsi, les performances de la communication peuvent être augmentées. De plus, si la régulation est effectuée dans la FPGA, le temps de cycle d'une boucle de régulation est fortement diminué par rapport à une régulation exécutée dans la CPU. A titre d'exemple, lors des essais, la boucle de régulation de l'entraînement électrique exécutée dans la CPU prend environ 400μs contre environ 50μs dans la FPGA. De plus, l'ajout de régulation en parallèle dans la CPU ralentit la cadence de toutes les boucles alors que dans la FPGA, ce phénomène n'est pas observé puisque les portes logiques sont activées en parallèle.

La figure 8.1 montre la structure de programmation de l'automate. Pour les raisons expliquées précédemment, la régulation est programmée dans la FPGA et la communication dans la CPU. Quatre boucles de régulations sont implémentées dans la FPGA. Deux entraînements électriques et deux systèmes instables à bille sont asservis.

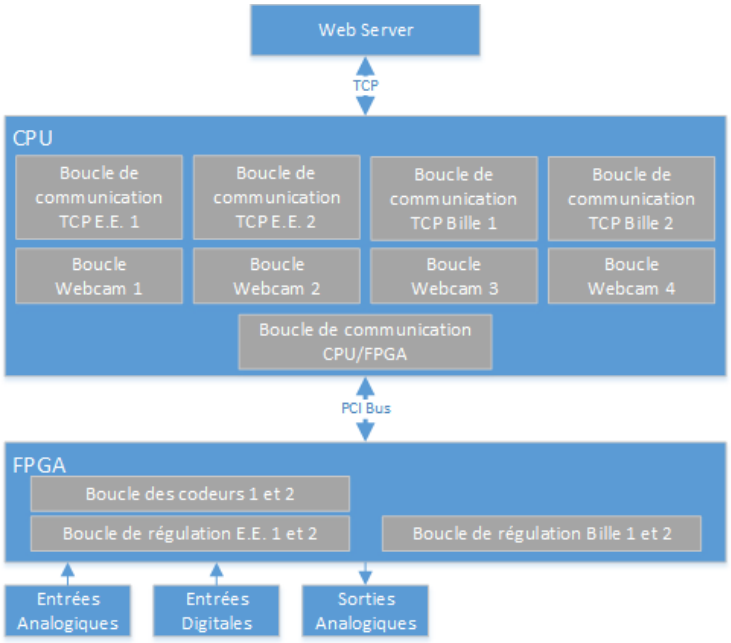


FIGURE 8.1 – Structure de la programmation en parallèle de 4 maquettes dans le cRIO

8.1 Programmation de la FPGA

Le programme de la FPGA contient l'algorithme de calcul de la vitesse et de la position des codeurs incrémentaux ainsi les deux boucles de régulation des quatre maquettes.

8.1.1 Algorithme des codeurs incrémentaux

L'algorithme des codeurs n'est pas cadencé. Le temps d'exécution de l'algorithme dépend ainsi de la fréquence des signaux dans le réseau de portes logiques de la FPGA. En pratique, le temps mesuré de cet algorithme est d'environ 18μs.

Les codeurs incrémentaux des entraînements électriques possèdent deux canaux (A et B) qui génèrent 2000 fronts par tour. La figure 8.2 permet de déterminer l'équation booléenne nécessaire au calcul de la position du codeur en fonction des fronts des deux canaux.

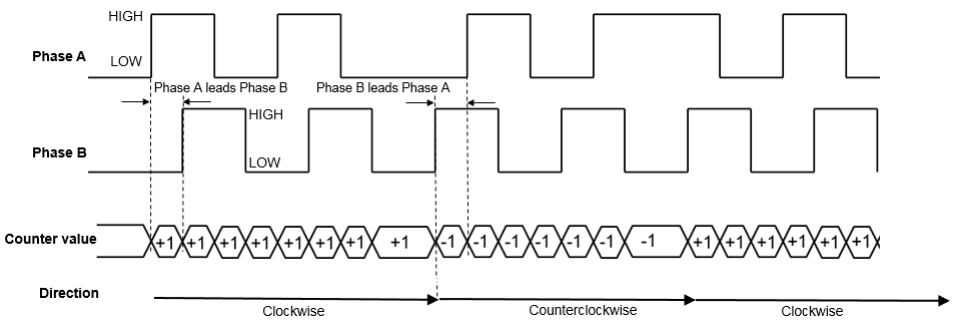


FIGURE 8.2 – Signaux d'un codeur incrémental à deux canaux
Source : `https://www.mathworks.com/help/supportpkg/freescalefrdmk64fboard/ref/quadratureencoder.html` (consulté le 07.08.2018)

Lorsqu'un front montant est détecté sur l'un des deux canaux, la position doit être incrémentée ou décrémentée d'une unité en fonction de l'état du canal opposé. Par exemple, si un front est détecté sur le canal B, le canal A détermine si le codeur tourne dans le sens horaire ou anti-horaire et inversement.

Il est ainsi possible de déterminer le moment d'activer le compteur (variable "count") et le sens de rotation (variable "dir"). Les tables de vérité et de Karnaugh de la figure 8.3 donnent l'état des variables "count" et "dir" en fonction de l'état actuel (A et B) et précédent (A'et B') des signaux du codeur. Les cases contenant la valeur "x" peuvent prendre n'importe quelle valeur. Elles représentent un état impossible ou un état dans lequel la valeur n'est pas importante.

B'	B	A'	A	count	dir
0	0	0	0	0	x
0	0	0	1	1	1
0	0	1	0	1	0
0	0	1	1	0	x
0	1	0	0	1	0
0	1	0	1	x	x
0	1	1	0	x	x
0	1	1	1	1	1
1	0	0	0	1	1
1	0	0	1	x	x
1	0	1	0	x	x
1	0	1	1	1	0
1	1	0	0	0	x
1	1	0	1	1	0
1	1	1	0	1	1
1	1	1	1	0	x

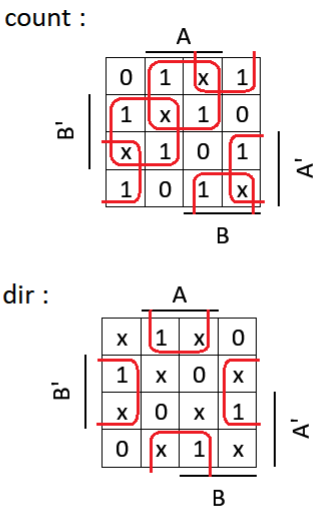


FIGURE 8.3 – Tables de vérité et Karnaugh du codeur

Les équations booléennes suivantes permettant de déterminer la position du compteur et le sens de rotation selon les deux tables ci-dessus.

$$count = A'\overline{A} + A\overline{A'} + B\overline{B'} + B'\overline{B} = A \oplus A' + B \oplus B' \tag{8.1}$$

$$dir = B'\overline{A} + A\overline{B'} = A \oplus B' \tag{8.2}$$

Pour déterminer la vitesse du codeur, il suffit de diviser la différence de deux mesures de position par leur intervalle de temps. La valeur du compteur est ainsi soustraite à son ancienne valeur toute les 500μs et divisée par ce même temps.
Le code LabVIEW implémenté dans la FPGA est représenté sur la figure 8.4.

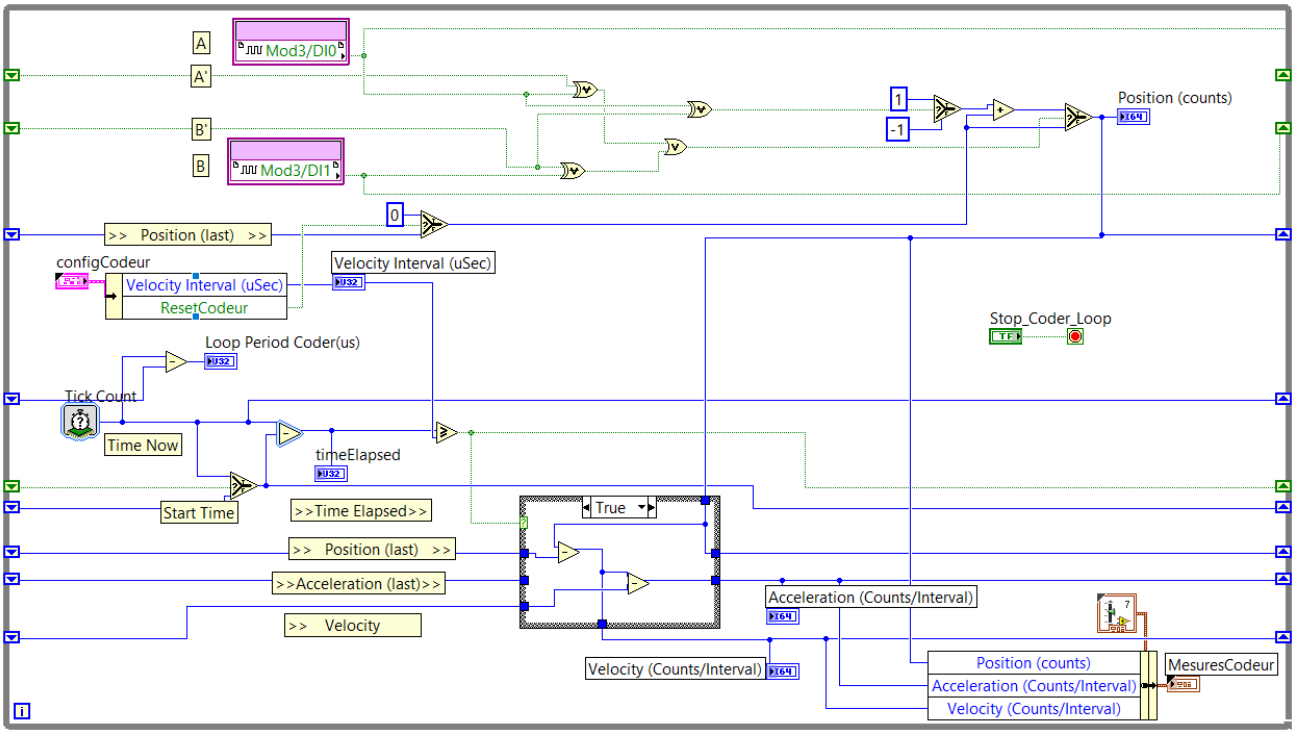


FIGURE 8.4 – Code FPGA de l'algorithme du codeur incrémental

8.1.2 Boucle de régulation de l'entraînement électrique

La maquette de l'entraînement électrique peut être pilotée en boucle ouverte ou en boucle fermée. En boucle fermée, elle peut être régulée à l'aide d'un PID, PI, PD, ou P + Feed-Forward. Le bloc PID utilisé est celui proposé dans la librairie de LabVIEW. Pour passer d'un régulateur à l'autre, les terme "Ki" ou "Kd" ou "Kff" sont simplement annulés. Un seul bloc PID est ainsi utilisé dans le programme. Le Feed-Forward est implémenté en ajoutant un régulateur P et un sommateur à la sortie du PID.

Un sélecteur à la sortie Feed-Forward est câblé afin de pouvoir passer du mode "Open-Loop" au mode "Closed-Loop". En boucle ouverte, la consigne est directement envoyée sur la sortie analogique. En boucle fermée, la consigne est soustraite à la mesure du capteur sélectionné. Le résultat est envoyé sur le bloc PID qui commande la sortie analogique. Juste avant d'envoyer le signal de commande sur la sortie analogique, le bloc fonction LabVIEW "In Range and Coerce" est utilisé afin de limiter la sortie à $\pm$10V.

Une structure conditionnelle est utilisée pour déterminer quelle entrée analogique lire en fonction du capteur sélectionné (tachymètre, potentiomètre, codeur vitesse, codeur position). Dans le cas du tachymètre, un filtre de Butterworth du $2^{ème}$ ordre, dont la fréquence de coupure vaut 500Hz (voir chapitre 7.2.1), est placé sur la mesure. Ce filtre est proposé dans la librairie LabVIEW "FPGA Math & Analysis".
Pour déterminer la position avec le potentiomètre, un calcul doit être effectué. En prenant le signal du capteur tel quel, la position ne peut être réglée que sur un tour puisque le potentiomètre fournit une tension entre 0 et 10V sur un tour.

8.1.3 Boucle de régulation du système instable à bille

Le code de la maquette à bille est à peu de chose près le même que celui de l'entraînement électrique. La seule différence est qu'il n'y a ici pas de structure conditionnelle pour le choix du capteur puisque la maquette n'en a qu'un. Le signal du capteur optique étant relativement "propre", aucun filtre n'est implémenté.

8.2 Programmation de la CPU

Le programme de la CPU contient principalement huit boucles de communication TCP avec le Web-Serveur dont quatre pour l'acquisition d'images de la webcam et quatre pour le transfert des mesures et commandes. La dernière boucle contient la communication entre la FPGA et la CPU.

8.2.1 Communication avec la FPGA

Une structure "Timed Loop" cadencée à 2ms est utilisée pour communiquer avec la FPGA. Les mesures et les commandes sont ainsi envoyées et reçues par intervalle de 2ms.
Le rôle de cette boucle est également de sauvegarder les mesures de la FPGA dans des tableaux de 125 valeurs. Ce sont ces tableaux de mesures qui sont envoyés au serveur. Un facteur de temps (x) permet de sauvegarder chaque point de mesure toutes les $x \cdot 2ms$. Ce facteur de temps peut prendre les valeurs prédéfinies 1, 5 ou 25. Ceci a pour but de pouvoir modifier l'échelle du graphique de l'interface utilisateur. Plus l'échelle de temps est grande, moins de points par seconde sont envoyés. Les paquets de mesures sont cependant toujours envoyés à la même fréquence. Par conséquent, ce facteur influe sur la taille du tableau de mesures. La taille du tableau en fonction du facteur de temps vaut 125/x et peut prendre ainsi les valeurs 125, 25 ou 5.
Dès lors qu'un client se connecte sur une maquette, une variable est incrémentée par pas de $x \cdot 2ms$. Celle-ci est également sauvegardée à la même fréquence que les mesures dans un tableau. Ce tableau, qui est également envoyé au client, permet de faire correspondre un temps à chaque valeur de mesure.
Ces tableaux sont ensuite convertis au format JSON (voir chapitre 9.3.1) avant d'être stockés dans des variables partagées.

La figure 8.5 illustre un exemple de code LabVIEW permettant de communiquer entre la CPU et la FPGA.

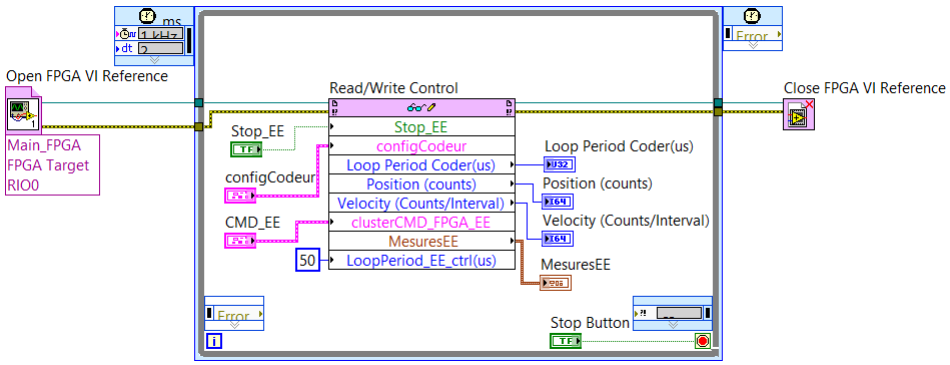


FIGURE 8.5 – Exemple de communication entre FPGA et CPU

Il faut tout d'abord ouvrir la référence au VI implémenté dans la FPGA. Le bloc "Read/Write Control" permet ensuite d'écrire sur les contrôles du VI de la FPGA et de lire ses indicateurs. La cadence de la "Timed Loop" définit le temps entre chaque échantillons lus et écrits sur la FPGA mais ne détermine pas la fréquence d'exécution de cette dernière. Ce qui signifie que la FPGA peut tourner beaucoup plus rapidement que la CPU.

8.2.2 Acquisition des images de la Webcam

Comme vu au chapitre 3.2, les Smart-Devices sont également composés d'une webcam. Les caméras employées sont des Logitech C270 (cf. annexe C) possédant une résolution maximale de 1280 x 720 pixels, ce qui est amplement suffisant dans le cadre des MOOLs.

Cette caméra a été choisie principalement pour son faible prix (29.10 Fr chez Distrelec) et également pour sa compatibilité avec le système d'exploitation Linux RT du cRIO-9067. National-Instrument ne fournit pas de liste des Webcams compatibles avec leurs automates mais garantit la compatibilité des webcams USB 2.0 compatibles UVC (USB Video Class). Ce standard décrit les webcams capables de diffuser de la vidéo en USB, sans nécessiter l'installation d'un driver.

L'acquisition d'images se fait à l'aide de la librairie LabVIEW "NI-IMAQdx" qui permet de récupérer les images de la webcam directement sous la forme d'une variable de type «image». La webcam possède plusieurs modes vidéo qui correspondent chacun à une certaine résolution et à un certain nombre d'images par seconde. Afin de privilégier la réactivité du contrôle et de la réception des signaux de mesures des différentes maquettes, le flux vidéo est limité à 10 images par secondes avec une résolution de 320x240 pixels. Le mode de la webcam correspondant à ce flux vidéo est le 112.

Avant d'être envoyée sur la page html, l'image doit être convertie en une chaîne de caractères de type "String". Cette opération est effectuée à l'aide de la fonction LabVIEW "IMAQ Flatten Image to String". Ce bloc fonction permet de convertir une donnée de type "image" en image JPEG sous la forme d'un "String" avec une certaine compression. La compression est fixée à 200/1000 ce qui permet une qualité acceptable tout en limitant la taille des paquets envoyés à la page html, 1000 étant la compression minimale et 0 maximale. Ce facteur de compression est fixé en le diminuant progressivement de 1000 à 200 jusqu'à atteindre une qualité d'image jugée raisonnable.

La figure 8.6 représente le code nécessaire à l'acquisition de l'image.

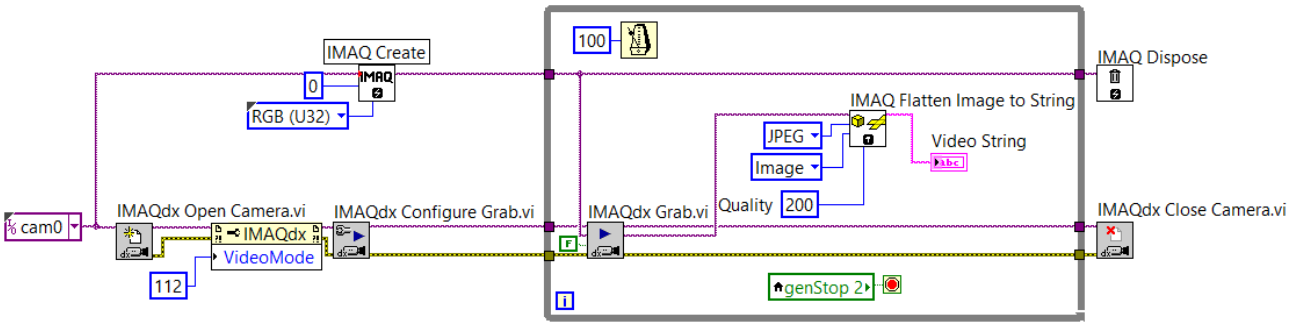


FIGURE 8.6 – Programmation de l'acquisition d'image

8.2.3 Communication avec le Web-Serveur

Comme vu au chapitre 5.2.4, le cRIO interagit avec le serveur à l'aide d'une communication TCP. La structure de base du programme permettant cela est celle de la figure 5.3. Deux VIs de communication par maquette sont utilisés, un pour la vidéo et l'autre pour les commandes et les mesures. Puisqu'il y a 4 maquettes, jusqu'à 8 communications peuvent tourner en parallèle. Ces communications se lancent dynamiquement avec le même principe que les VIs du serveur destinés au WebSocket (voir chapitre 6.1). Lorsque le serveur ouvre une communication, il envoie l'ID de la maquette sur laquelle il veut se connecter. Ainsi, avant de démarrer les deux VI de communication, le chemin des variables partagées est déterminé en fonction de cet ID. Ces variables permettent de transmettre les données entre la boucle de communication avec la FPGA et les boucles TCP. L'identifiant de la caméra est également déterminé à ce moment-là. Lorsqu'aucun client n'est connecté au Smart-Dervice, toutes les connexions de l'automate sont fermées.

Ce lancement dynamique de communications est effectué grâce au programme visible sur la figure 8.7.

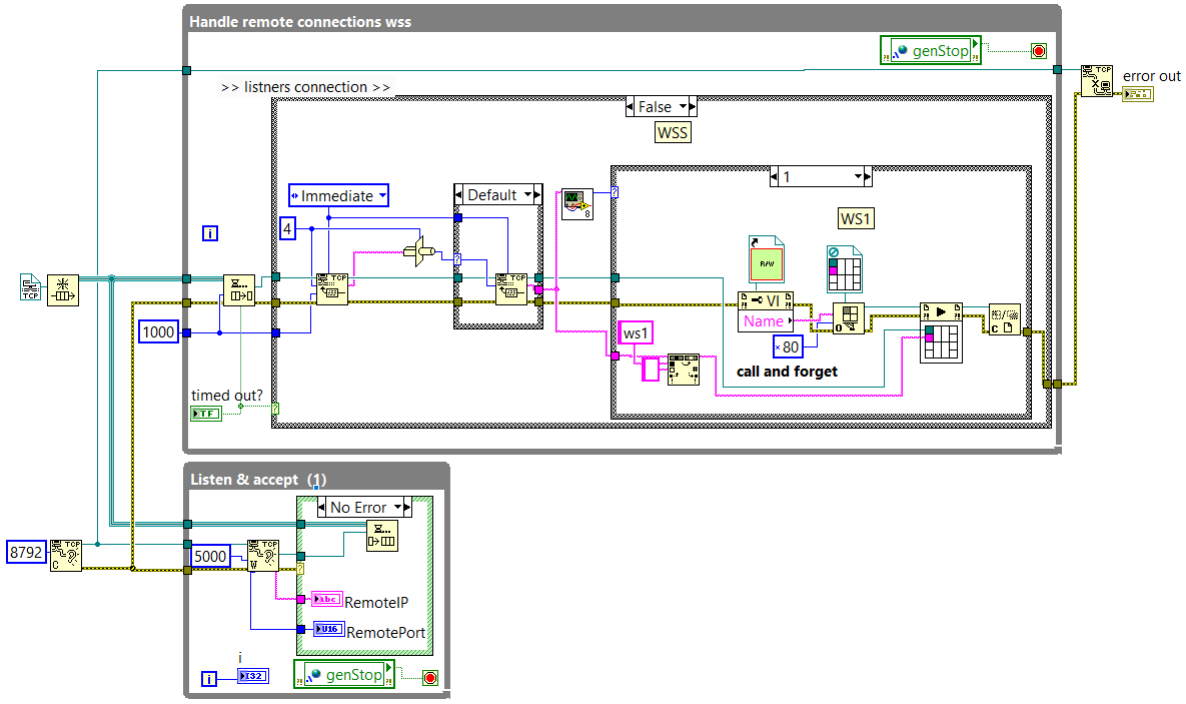


FIGURE 8.7 – Lancement dynamique des communications TCP

Les mesures sont envoyées au serveur par intervalle de 250ms et les commandes sont reçues toutes les 250ms également.

9 Interface Client

L'interface utilisateur du client permet de piloter et de visualiser les différentes expérimentations du laboratoire. Elle doit contenir les différents paramètres nécessaires à la régulation des maquettes. Un graphique temporel permet de visualiser les signaux de mesures et de contrôles. Les échelles du graphique peuvent être modifiées par l'utilisateur à l'aide de listes déroulantes dont les valeurs sont prédéfinies. De plus, un bouton "pause" permet de figer l'image du graphique. L'image de la webcam est également présente sur l'interface afin de rajouter du réalisme à l'expérience.
Les paramètres des différents régulateurs calculés dans le chapitre 7 sont définis comme valeurs par défaut sur l'interface. D'autres exemples d'interface client (Open-Loop et Closed-Loop) pour les maquettes de moteur DC et pour le système à bille sont présentés dans les annexes G à I.

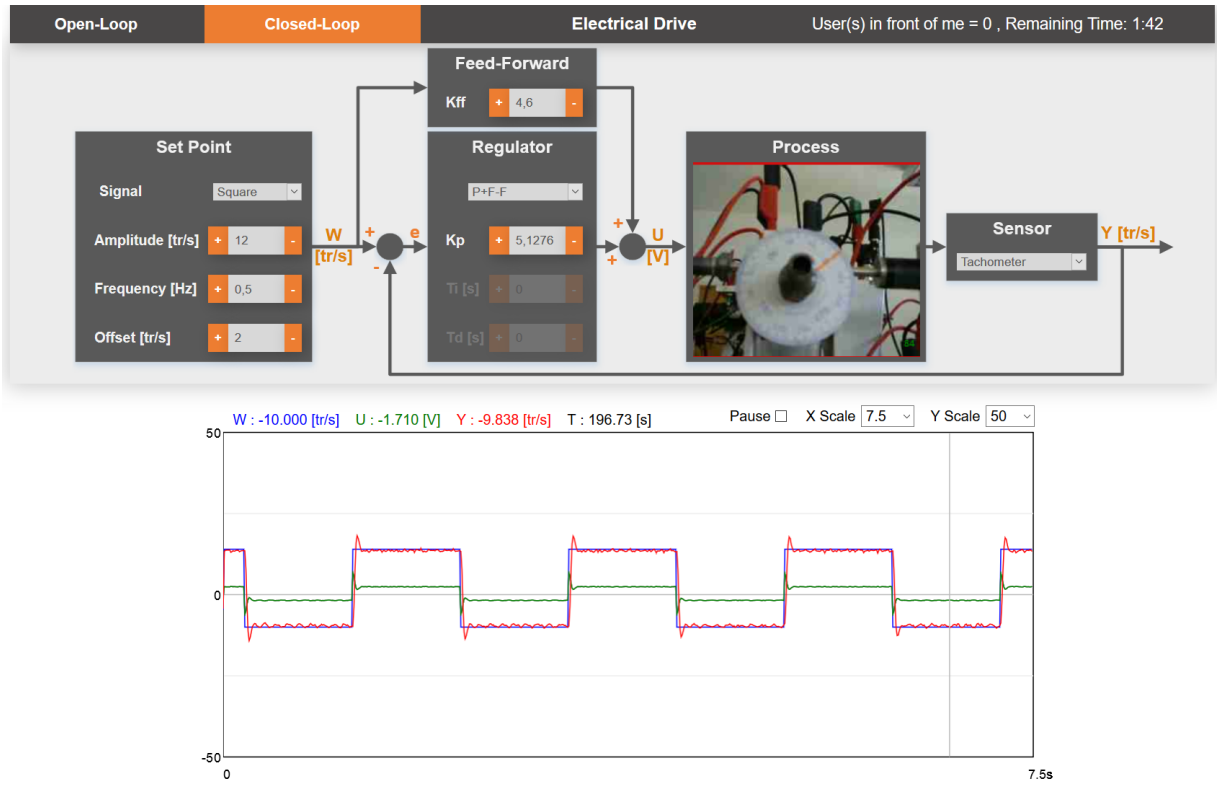


FIGURE 9.1 – Interface utilisateur pour la régulation du moteur DC

L'interface est développée en HTML5 afin de maximiser la compatibilité avec les ordinateurs des étudiants, qui tournent potentiellement sous différents systèmes d'exploitation (Mac OS, Windows 7, Windows 10, etc.). En effet, le code d'une page Web ne dépend pas du système d'exploitation sur lequel il est exécuté. Seul un navigateur Web à jour est requis afin de garantir la compatibilité avec le plus grand nombre d'appareils.
HTML5 (HyperText Markut Language 5) est le langage de programmation utilisé pour le développement de pages Web.
« *HTML5 (HyperText Markup Language 5) est la dernière révision majeure du HTML (format de données conçu pour représenter les pages web). Dans le langage courant, HTML5 désigne souvent un ensemble de technologies Web (HTML5, CSS3 et JavaScript) permettant notamment le développement d'applications.* [4] ».

4. Wikipedia, https ://fr.wikipedia.org/wiki/HTML5 (Consulté le 04.08.2018)

Ce type de programmation ne nécessite pas de programme de développement particulier. N'importe quel éditeur de texte peut être utilisé.
Trois langages différents sont utilisés pour développer la page Web : HTML, CSS, et JavaScript.

9.1 HTML

La partie HML de la page Web contient la déclaration des éléments qui figurent sur la page. Les différentes fonctionnalités sont appelées à l'aide de balises. Ces dernières permettent de définir des objets tels que des boutons, du texte, des images, etc.

Les balises utilisées pour l'interface sont les suivantes :

- : Balise générique de type block
- <button></button> : Bouton
- <select></select> : Liste déroulante
- <option></option> : Elément d'une liste déroulante
- <input type="number"></input> : Champ de saisie d'un nombre
- <canvas></canvas> : Zone de dessin

La structure de base pour programmer une page Web est présentée dans la figure 9.2.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Titre</title>
    </head>

    <body>

    </body>
</html>
```

FIGURE 9.2 – Structure de base d'une page HTML

Le code compris entre les balises "head" contient la configuration de la page. Cette partie est appelée "en-tête" de la page Web. Le code s'y trouvant n'est pas visible sur la page. Tout ce qui est déclaré entre les balises "body" (corps de la page) est visible sur la page. C'est là que se trouve les différents boutons, affichages et indicateurs de l'interface.

La balise "div" permet de définir des zones rectangulaires sur la page. Ces zones peuvent avoir des dimensions définies en nombre de pixels ou être adaptatives à la dimension de l'écran. Ces blocs sont utilisés pour structurer la page en déterminant la position et la taille des éléments s'y trouvant.

La figure 9.3 décrit la structure globale de l'interface.

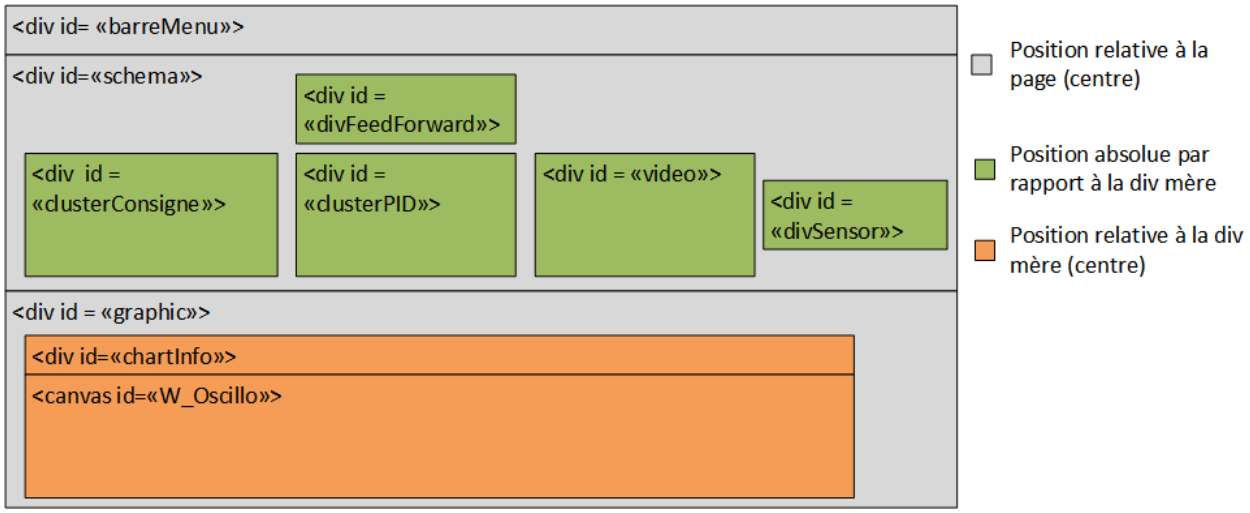


FIGURE 9.3 – Structure de la page Web

La mise en forme et la position de ces blocs sont programmées dans la partie CSS.
Des attributs peuvent être ajoutés aux différentes balises. Cela permet de définir par exemple une classe et un ID dans les divisions pour faire référence à ces dernières depuis le code CSS. L'attribut "ID" est utilisé si on souhaite faite référence à une balise dans le CSS ou le Javascript. Ceci dans le but d'associer une action ou de modifier l'aspect de cette dernière. L'attribut "class", contrairement à l'attribut "ID", peut être utilisé pour plusieurs balises. On peut ainsi affecter les mêmes propriétés à toute une série de balises avec une seule déclaration.

La figure 9.4 représente la structure détaillée des sous-blocs de la figure 9.3.

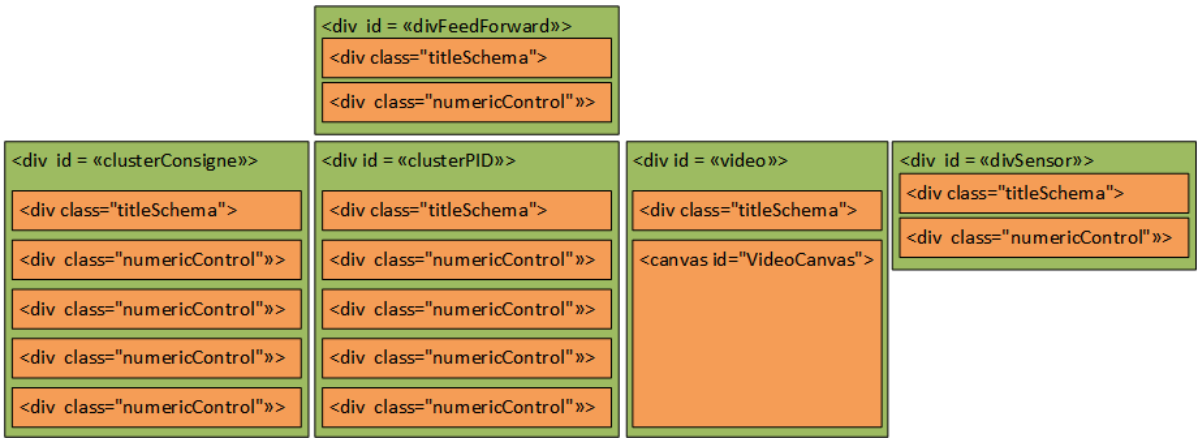


FIGURE 9.4 – Structure des blocs de la page Web

L'exemple suivant illustre la manière d'implémenter la structure dans le code html.

```
<body>
    <div id="schema">
        <div id="clusterConsigne">
            <div class="titleSchema">Set Point</div>
            <div class="numericControl">
            </div>
        </div>
        <div id="clusterPID">
            <div class="titleSchema">Regulator</div>
            <div class="numericControl">
            </div>
        </div>
    </div>
</body>
```

FIGURE 9.5 – Exemple de structure HML

Les différents boutons et champs de contrôle sont définis dans les divisions de la classe "numericControl". Cela permet de définir dans le CSS une seule et même mise en page pour les contrôles. Cette division est composée d'un texte de description, d'un bouton pour incrémenter la valeur d'une variable, d'un autre pour la décrémenter et d'une entrée numérique. Ces quatre éléments sont visibles sur la figure suivante :

FIGURE 9.6 – Exemple d'une division "numericControl"

Le code qui définit cette structure est visible sur la figure 9.7 :

```
<div class="numericControl">
    <text class="idParam" id="ampText">Amplitude [V]</text>
    <button class="addBtn" >+</button>
    <input type="number" class="numberField" id="ampValue" ></input>
    <button class="subBtn" >-</button>
</div>
```

FIGURE 9.7 – Exemple de code html d'un "numericControl"

Les blocs de la classe "titleSchema" contiennent le titre de chaque sous-division. Leur mise en forme est identique pour toutes les divisions.

Le dernier élément HTML utilisé est la balise "select". Cette balise définit une liste déroulante permettant à l'utilisateur de sélectionner des valeurs contenues dans cette dernière. Cet élément est utilisé pour sélectionner la forme du signal de consigne, le type de régulateur et le type de capteur (dans le cas de l'entraînement électrique). Les valeurs de la liste sont définies à l'aide de la balise "option".

L'implémentation de cet élément est faite de la manière suivante :

```
<div class="numericControl">
    <text class="idParam">Signal</text>
    <select name="signalShape" id="signalShape">
        <option value="zero">Zero</option>
        <option value="constant">Constant</option>
        <option value="square">Square</option>
        <option value="sinus">Sinus</option>
        <option value="triangle">Triangle</option>
        <option value="prbs">Prbs</option>
    </select>
</div>
```

FIGURE 9.8 – Code html de la liste de signaux

Pour affecter une fonction JavaScript à une action faite sur un bouton ou sur un champ d'entrée par exemple, le nom de la fonction doit être déclaré en attribut dans la balise de l'élément. Cela permet, entre-autre, d'envoyer les paramètres au Web Serveur lorsqu'on change une valeur sur l'interface. Pour ce faire, les attributs "onclick" pour les boutons et "onchange" pour les champs d'entrée sont utilisés. L'exemple suivant définit un champ de type "number" appelant la fonction "setPValue" lors d'un changement de sa valeur.

```
<input type="number" class="numberField" id="tiValue" onchange="setPValue('tiValue')"></input>
```

FIGURE 9.9 – Appel d'une fonction JavaScript au changement d'une valeur d'entrée

9.2 CSS

Le CSS est la partie du code de la page Web permettant de mettre en forme les éléments déclarés dans la partie html. Ce code est écrit dans l'en-tête de la page compris entre les balises "<style></style>".
La syntaxe pour appeler un objet possédant un "ID" est la suivante : $\#id\{...\}$. Si on souhaite appeler une classe plutôt qu'un "ID", le dièse est remplacé par un point : $.class\{...\}$.
Les différents arguments permettant de mettre en forme l'objet appelé sont déclarés entre les accolades.

La position des divisions est gérée à l'aide des sept arguments suivants :

- position
- top
- right
- align-items
- display
- justify-content
- flex-wrap

9.2.1 Position du "parent"

L'argument "position" peut prendre les valeurs "relative" ou "absolute". Si la valeur est "relative", la position de l'objet dépend de la division mère. Au contraire, si la valeur est "absolute", la position de l'objet est gérée par les arguments "top" et "right" qui définissent l'espace en haut et droite de la division "fille" par rapport à la "mère". Cette distance peut être définie en nombre de pixels ou en pourcentage de la taille du parent.

9.2.2 Position des "enfants"

La propriété "align-items" accepte entre-autre les valeurs "flex-start, flex-end et center". Cette propriété permet positionner verticalement les objets appartenant à une division respectivement en haut, en bas et au centre.
La propriété "display" est utilisée avec la valeur "flex". Cela permet de distribuer horizontalement et verticalement les objets d'une division parente. Cette distribution est gérée avec l'argument "justify-content" qui peut prendre les valeurs suivantes :

- flex-start : Place l'élément au début du parent (horizontalement)
- center : Place l'élément au centre du parent (horizontalement)
- flex-end : Place l'élément à la fin du parent (horizontalement)
- space-between : Distribue les éléments horizontalement dans le parent

Avec la valeur "wrap" associée à la propriété "flex-wrap", les éléments sont renvoyés à la ligne ce qui permet de les distribuer verticalement. Ceci est utilisé dans les divisions de la consigne et du régulateur pour renvoyer les différents contrôles à ligne.

L'extrait de code suivant définit la position du "clusterConsigne" (fig 9.3) dans la division "schema" ainsi que la position de ses enfants.

```
#clusterConsigne{
    position: absolute;
    top: 90px;
    left: 12px;

    align-items: flex-start;
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
}
```

Position de la division

Position des enfants

FIGURE 9.10 – Paramètres CSS de positionnement de "clusterConsigne"

Les dernières propriétés utilisées dans le CSS sont celles permettant de définir les dimensions, la couleur de fond et la couleur du texte des différents éléments.

- width : Largeur de l'élément en pixels
- height : Hauteur de l'élément en pixels
- backgroud-color : Couleur de fond de l'élément
- color : Couleur du texte de l'élément

Les propriétés "width" et "height" définissent les dimensions d'un élément. Si ces dernières ne sont pas appelées dans la configuration, les dimensions de l'objet s'adaptent en fonction de ce qu'il contient. Par exemple, s'il contient une division de x par y pixels, ses dimensions seront de x par y pixels.
Les deux paramètres de couleurs prennent comme valeur une couleur au format hexadécimal. Ce format décrit les couleurs avec le système rgb (couleurs rouge, verte et bleue, codées sur un octet). La fonction "rgb(r,g,b)" permet d'effectuer ce formatage en lui donnant comme argument la valeur des trois couleur entre 0 et 255.

9.3 JavaScript

La dernière partie de la programmation est le JavaScript. Cette section est la plus conséquente de la programmation Web. Elle contient les différentes fonctions permettant de communiquer avec le serveur et l'affichage des différents signaux de mesures sur le graphique. Le code JavaScript se trouve dans l'en-tête de la page html entre les balises <script></script>.

Les différentes fonctions sont déclarées avec la syntaxe suivante : "$function\ myFunction(argmuent)\{...\}$". Les variables peuvent être déclarées soit en global, soit en local à l'intérieur d'une fonction. La déclaration se fait de la manière suivante : "$var\ myVariable$;". Les types de variable utilisés dans ce programme sont "Number", "String" et "Objet". En JavaScript, les nombres sont des flottants signés encodés sur 64 bits. Le type de la variable est assigné lorsque celle-ci reçoit sa première valeur.

Toutes les parties de la programmation sur la communication WebSocket ainsi que le graphique sont tirées du code fourni par l'EPFL et développé pour leurs laboratoires à distance.

9.3.1 JSON

« *JSON (JavaScript Object Notation) est un format d'échange de données léger. Il est facile pour les humains de le lire et de l'écrire. Il est facile pour les machines de l'analyser et de le générer. Il est basé sur un sous-ensemble du langage de programmation JavaScript , la norme ECMA-262 3ème édition - décembre 1999 . JSON est un format de texte totalement indépendant du langage, mais utilise des conventions familières aux programmeurs de la famille des langages C, notamment C, C ++, C #, Java, JavaScript, Perl, Python et bien d'autres. Ces propriétés font de JSON un langage d'échange de données idéal.*

JSON est construit sur deux structures :

- *Une collection de paires nom / valeur. Dans divers langages, ceci est réalisé sous la forme d'un objet , d'un enregistrement, d'une structure, d'un dictionnaire, d'une table de hachage, d'une liste de clés ou d'un tableau associatif.*
- *Une liste ordonnée de valeurs. Dans la plupart des langues, ceci est réalisé sous la forme d'un tableau, d'un vecteur, d'une liste ou d'une séquence.*

Ce sont des structures de données universelles. Pratiquement tous les langages de programmation modernes les prennent en charge sous une forme ou une autre. Il est logique qu'un format de données interchangeables avec les langages de programmation repose également sur ces structures.

En JSON, ils prennent ces formes

- *Un objet est un ensemble non ordonné de paires nom / valeur.*
- *Un objet commence par { (accolade gauche) et se termine par } (accolade droite).*
- *Chaque nom est suivi par : (deux-points) et le nom / paires de valeurs sont séparées par une virgule.*

[5] »

5. www.json.org (consulté le 06.08.2018)

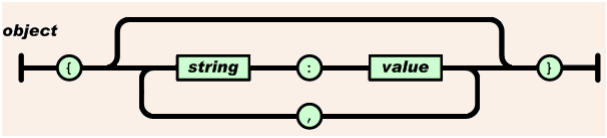


FIGURE 9.11 – Structure d'un objet JSON (Source : www.json.org (consulté le 06.08.2018)

« *Un tableau est une collection ordonnée de valeurs. Un tableau commence par [(crochet gauche) et se termine par] (crochet droit) . Les valeurs sont séparées par , (virgule).* [6]»

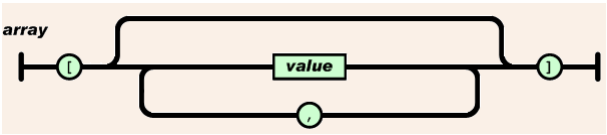


FIGURE 9.12 – Structure d'un tableau JSON (Source : www.json.org (consulté le 06.08.2018))

La figure 9.13 est un exemple de mise en forme d'un message JSON. En JavaScript, cette mise en forme se fait très facilement à l'aide de la méthode JSON.stringify().

```
var sensorRequest = {
    method: 'getSensorData',
    sensorId: 'Video',
    accessRole: 'controller'
    }
var jsonRequest = JSON.stringify(sensorRequest);
```

Names — Values

FIGURE 9.13 – Mise en forme d'un message JSON en JavaScript

6. www.json.org (consulté le 06.08.2018)

9.3.2 Initialisation

A l'ouverture de la page Web, la fonction "setup()" est appelée. Pour ce faire, la fonction est assignée à l'argument "onload" dans la déclaration de la balise "body".

```
<body onload="setup();" onbeforeunload="Close();">
```

FIGURE 9.14 – Initialisation de la page Web

Cette fonction permet d'initialiser la communication avec le serveur ainsi que d'afficher les différents éléments sur la page Web. La séquence d'exécution de cette dernière est visible sur la figure 9.15.

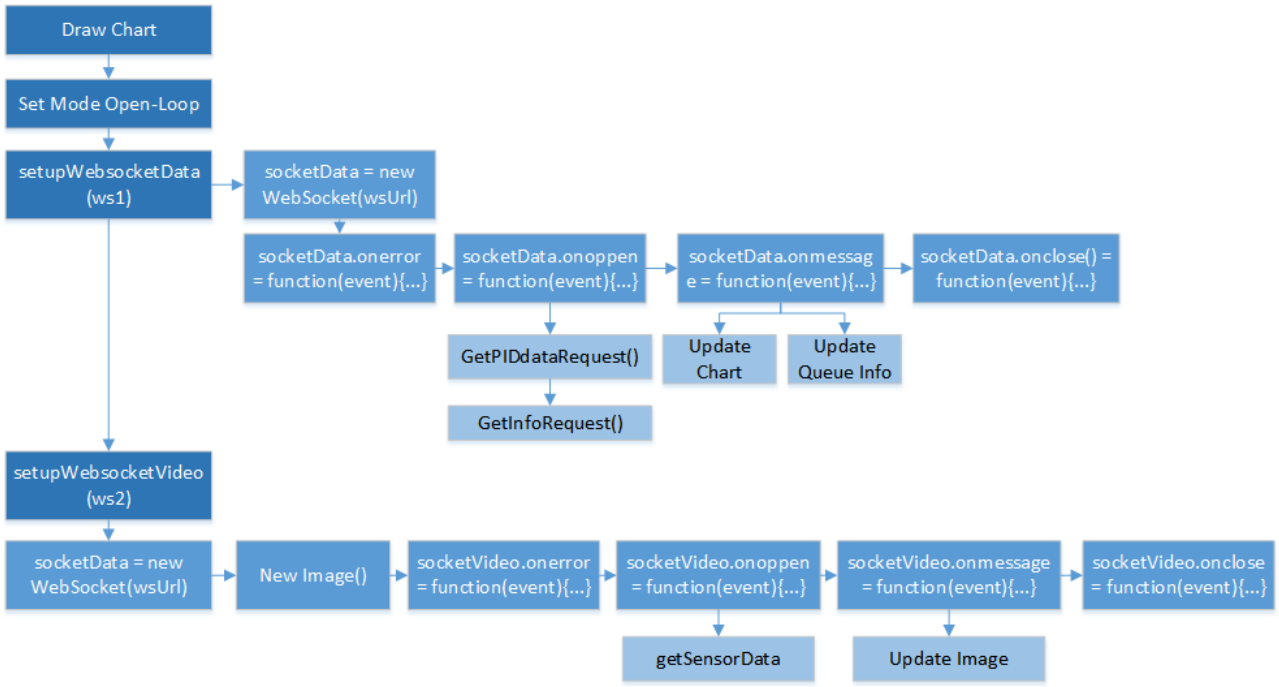


FIGURE 9.15 – Séquence d'ouverture de la page Web

Tout d'abord, le "canvas" du graphique permettant d'afficher les mesures est dessiné, suivit par le schéma de régulation visible sur la figure 9.1.
Ensuite, la connexion pour l'échange des signaux de mesures et de commandes est initialisée. Pour ce faire, l'objet "socketData" de la classe "WebSocket" est instancié avec l'url de communication (voir chapitre 5.3.3). Des évènements sont ensuite créés pour les fonctions "onerror", "onopen", "onmessage" et "onlose". Ces fonctions appartiennent à la classe "WebSocket" et sont ainsi appelées à chaque évènement de l'objet "socketData".
Lorsque la communication est ouverte, la fonction "onpen" est exécutée. Elle envoie alors au serveur deux requêtes au format JSON. Ces requêtes demandent au serveur d'envoyer les mesures du Smart-Device ainsi que la position dans la file d'attente et le temps restant avant d'être déconnecté dans le cas où le client est contrôleur (voir chapitre 6.2).
Quand un message est reçu, la fonction "onmessage" est appelée. Si le message contient les mesures du Smart-Device, elles sont enregistrées sous forme de tableau dans une variable local et mise à jour sur le graphique. Si ce sont les informations de position dans la file d'attente et de temps restant, les champs les concernant sont mis à jour sur la page Web.
Finalement, lorsque la communication est fermée, la fonction "onclose" est appelée. Celle-ci ne fait qu'afficher dans la console du navigateur que la communication a été fermée.

Tout d'abord, une variable ("myImage") est déclarée avec le type Image. La communication de la Video est ensuite initialisée. La même procédure que pour les "datas" est suivie. L'objet "socketVideo" de la classe WebSocket est instanciée avec l'url de la vidéo (voir chapitre 5.3.3). Une requête est alors envoyée au serveur afin de recevoir les images de la maquette à l'aide de la méthode "onopen". Lorsque les images sont reçues, la méthode "onmessage" est appelée. Celle-ci transforme le message reçu sous la forme d'une chaîne de caractère en image à l'aide le fonction "URL.createObjectURL(msg)". Cette image est ensuite mise à jour sur la page Web.
A la fermeture de la communication, la fonction "onclose" affiche simplement dans la console du navigateur que la connexion est fermée.

9.3.3 Programme en fonctionnement

Une fois la procédure d'initialisation achevée, le JavaScript gère les différentes entrées de l'interface. Lorsqu'un champ est modifié ou "cliqué", la fonction "sendActuatorChange()" est appelée. Celle-ci crée un message JSON avec toutes les valeurs de commandes actuelles et l'envoie au serveur à l'aide de la méthode "send" de l'objet "socketData" vu au chapitre 9.3.2.

Lorsque l'utilisateur change la valeur de ces champs, le JavaScript vérifie que la valeur introduite est comprise entre des bornes prédéfinies. Cela empêche l'utilisateur d'introduire par exemple une valeur trop élevée pour la consigne du moteur. Ceci est implémenté pour tous les champs de la page Web.

Le figure 9.16 décrit le fonctionnement du programme en fonctionnement.

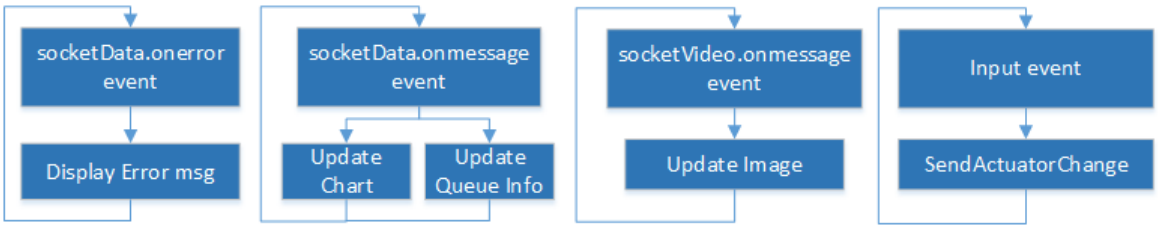


FIGURE 9.16 – Diagramme d'exécution du programme en fonctionnement

Si la communication avec le serveur est coupée pour une quelconque raison, la fonction "onerror" est appelée et affiche un message d'erreur sur l'interface.

9.3.4 Fermeture

La fermeture de la page Web est gérée grâce à la fonction "Close()". Cette dernière est appelée juste avant que la page ne soit fermée. Pour ce faire, la fonction est associée à attribut "onbeforeunload" visible sur la figure 9.15.

Le code qu'elle exécute permet de fermer les deux communications WebSocket (socketData et socketVideo) à l'aide de la méthode "close" de la classe "WebSocket".

10 Tests de l'infrastructure

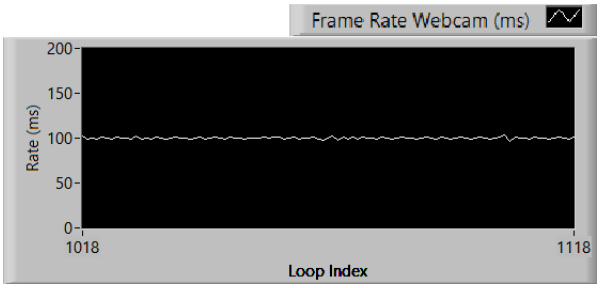
Les trois différentes parties de l'infrastructure ne pouvant fonctionner séparément (sauf pour le cRIO), toute la chaîne est testée. L'infrastructure des MOOLs de la HES-SO Valais, possède actuellement un automate en réseau et quatre maquettes connectées sur ce dernier.

10.1 Mesures du temps entre les paquets

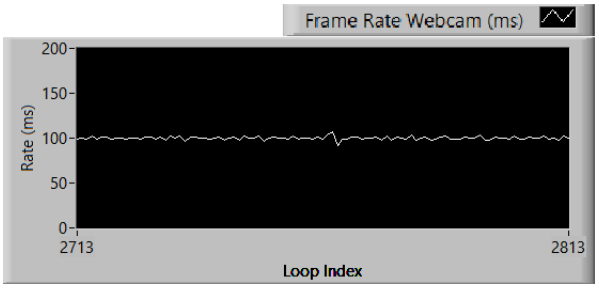
Dans un premier temps, un seul client est connecté sur une des maquettes. Le temps entre les paquets de mesures est mesuré au niveau du serveur et du client. Le tableau contenant l'instant ou chaque point de mesure a été enregistré, permet de déterminer si des paquets ne sont pas transmis dans le cas où l'intervalle de temps est trop grand entre deux paquets. De plus, le temps entre chaque "frame vidéo" est affiché dans la console du navigateur de la page web. Toutes les entrées de la page web sont vérifiées en introduisant une valeur et en vérifiant que celle-ci soit bien transmise au cRIO. Pour cela, les valeurs envoyées à la FPGA sont visualisées sur le VI principal de l'automate.

D'après les tests, les paquets sont reçus sans pertes et à la bonne fréquence.

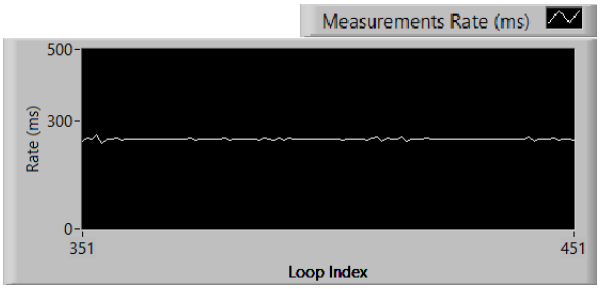
Un essai est ensuite effectué avec les quatre maquettes tournant en même temps. Les mêmes valeurs que pour l'essai précédent sont relevées. Tout comme pour le premier essai, aucune latence n'est observée avec les mesures et la vidéo. Les graphiques de la figure 10.1 indiquent le temps entre chaque paquet de mesures et vidéo, pour une maquette active et pour quatre.



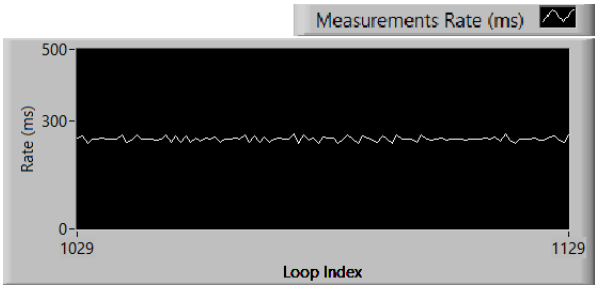
(a) 1 maquette active (vidéo)



(b) 4 maquette active (vidéo)



(c) 1 maquettes actives (mesures)



(d) 4 maquettes actives (mesures)

FIGURE 10.1 – Mesure du temps entre les paquets des mesures et de la vidéo

Le temps maximal observé entre deux "frames" pour une maquette active est de 105ms pour la vidéo et de 260ms pour les mesures. Pour quatre maquettes, le temps maximal observé est de 110ms pour la vidéo et de 265ms pour les mesures. Ce n'est pas un problème si de temps à autre un paquet prend plus de temps pour arriver puisque celui qui le suit rattrape le retard. De plus le dépassement maximal est relativement faible (15ms pour les mesures et 10ms pour la vidéo) et n'est pas perceptible sur la visualisation.

10.2 Mesures de l'utilisation de la CPU du cRIO

L'utilisation du processeur de l'automate est relevée à l'aide du "Gestionnaire de systèmes distribués NI". Cette utilisation est représentée sur la figure 10.2. Quatre communications sont ouvertes les unes après les autres sur les quatre différentes maquettes.

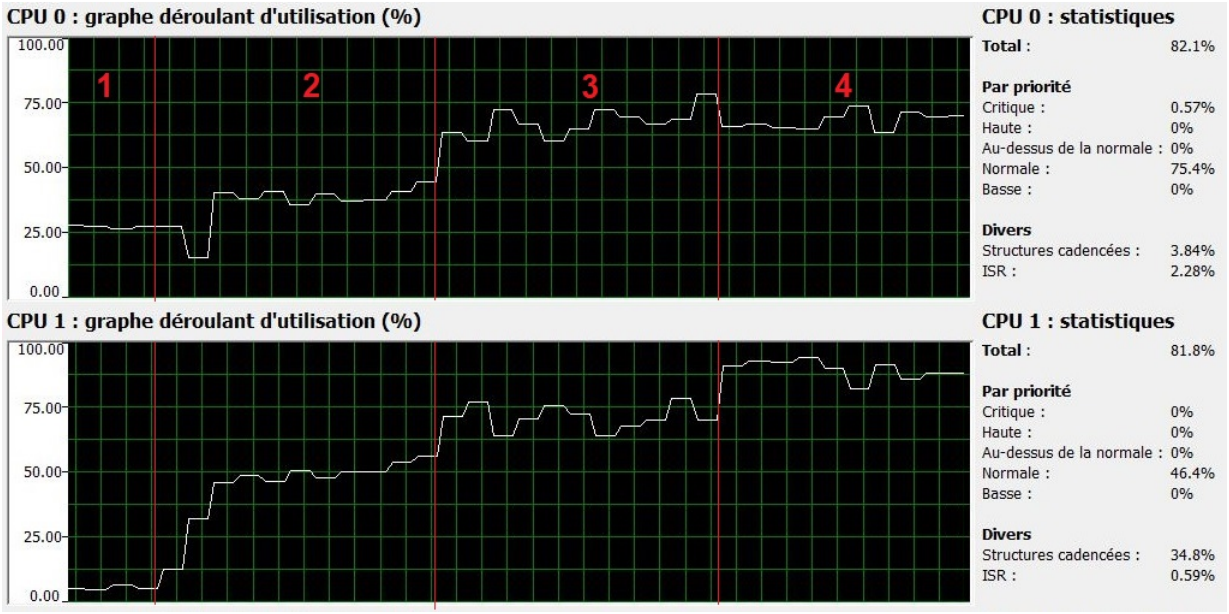


FIGURE 10.2 – Utilisation de la CPU de l'automate avec 4 connexions

A l'ouverture de la quatrième connexion, le "CPU 1" atteint environ 93% d'utilisation et le "CPU 0" reste constant à environ 70%. D'après ces mesures, il est possible d'en conclure que le nombre maximal de maquettes par automate est certainement de quatre avant de saturer la CPU.

10.3 Mesures de l'utilisation réseau et processeur de la machine virtuelle

Du côté de la machine virtuelle, l'utilisation du processeur et du réseau sont également relevées. Ces données sont visibles sur la figure 10.3.

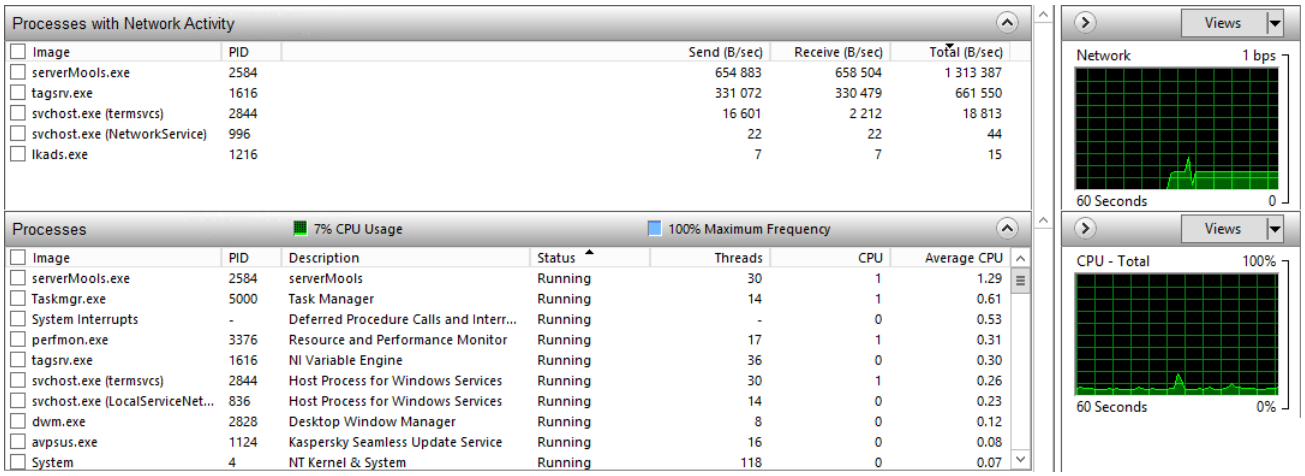


FIGURE 10.3 – Utilisation réseau et processeur de la machine virtuelle

L'application LabVIEW ("serverMools.exe") utilise 1.29% de la capacité de la CPU avec quatre utilisateurs connectés. Il faut ajouter en plus l'application "tagsrv.exe" qui sert à gérer les variables partagées et qui consomme 0.3% de la CPU.
Un dernier test est effectué avec 20 clients connectés par maquettes, soit un total de 80 clients. Les pages web sont toutes ouvertes sur le même PC. Ceci a pour effet de fausser les mesures de réactivité, puisque 80 pages sont ouvertes dans le même navigateur, ce qui a pour effet de le ralentir. Cependant, cela permet de vérifier le bon fonctionnement des files d'attentes des différentes maquettes.
D'après cet essai, à chaque nouvelle ouverture de page Web, le serveur fait bien le lien avec la maquette ayant le moins de client.

11 Conclusion et perspectives

L'objectif de ce travail de diplôme, en collaboration avec l'EPFL, est d'étendre le principe des MOOCs aux Laboratoires de Mécatronique. Pour atteindre cet objectif, une infrastructure de type serveur-client a été développée. Une interface utilisateur, sous la forme d'une page web, permet d'accéder aux expériences de laboratoire du cours de Mécatronique.

A l'heure où sont écrites ces lignes, deux types d'expérimentations sont intégrées à l'infrastructure du MOOLs. La première expérimentation est un entraînement électrique composé d'un moteur DC asservi en vitesse et position. La deuxième est une maquette de régulation de système instable à bille. Leurs pages Web de contrôle sont disponibles sur la plateforme Moodle dédiée aux MOOLs de la HES-SO Valais (Lien : `https://cyberlearn.hes-so.ch/course/view.php?id=11754`). Cette interface permet de piloter les maquettes en "Open-Loop" ou en "Closed-Loop". Dans le deuxième cas, plusieurs types de régulateur peuvent être sélectionnés. Par la suite, d'autres expérimentations devraient être intégrées à l'infrastructure. Concernant la maquette à bille, sa régulation devrait être améliorée par l'ajout d'un prédicteur de Smith (voir chap. 7.6.3).

L'accès au laboratoire se fait au travers d'un Web-Serveur qui relaie les informations entre les clients et les maquettes. Ces dernières sont pilotées à l'aide d'un automate industriel National Instrument programmé en LabVIEW.

L'interface utilisateur est convivial et permet de commander et de visualiser les différents signaux des expériences. De plus, un retour vidéo ajoute du réalisme aux manipulations.

Plusieurs étudiants peuvent se connecter en même temps sur les différents postes. Pour gérer ce phénomène, un seul étudiant a le contrôle de la maquette lorsque les autres ne peuvent que l'observer. Le premier a un temps limite de contrôle afin de libérer la place aux autres utilisateurs.

Tous les points du cahier des charges ont été remplis.

Il reste à présent à vérifier le bon fonctionnement et la réactivité de l'installation lorsque beaucoup d'étudiants se connectent en même temps. Un autre objectif est d'augmenter progressivement le nombre d'expérimentations dans le parc. De plus, une base de données devrait être créée afin de sauvegarder les manipulations des étudiants. Celle-ci pourrait être exploitée par les professeurs pour effectuer des statistiques et tirer des conclusions afin d'améliorer leurs travaux pratiques et d'ajouter d'autres fonctionnalités pédagogiques au système MOOLs. De plus, la base de données pourrait être utilisée par les étudiants pour comparer leurs travaux avec ceux de leurs collègues afin d'optimiser leurs performances.

Date et Signature

Sion, le 16 août 2018

Thomas Moniquet

Références

[1] Olivier Bonaventure. *Computer Networking : Principles, Protocols and Practice (Release 0.25)*, 2014.

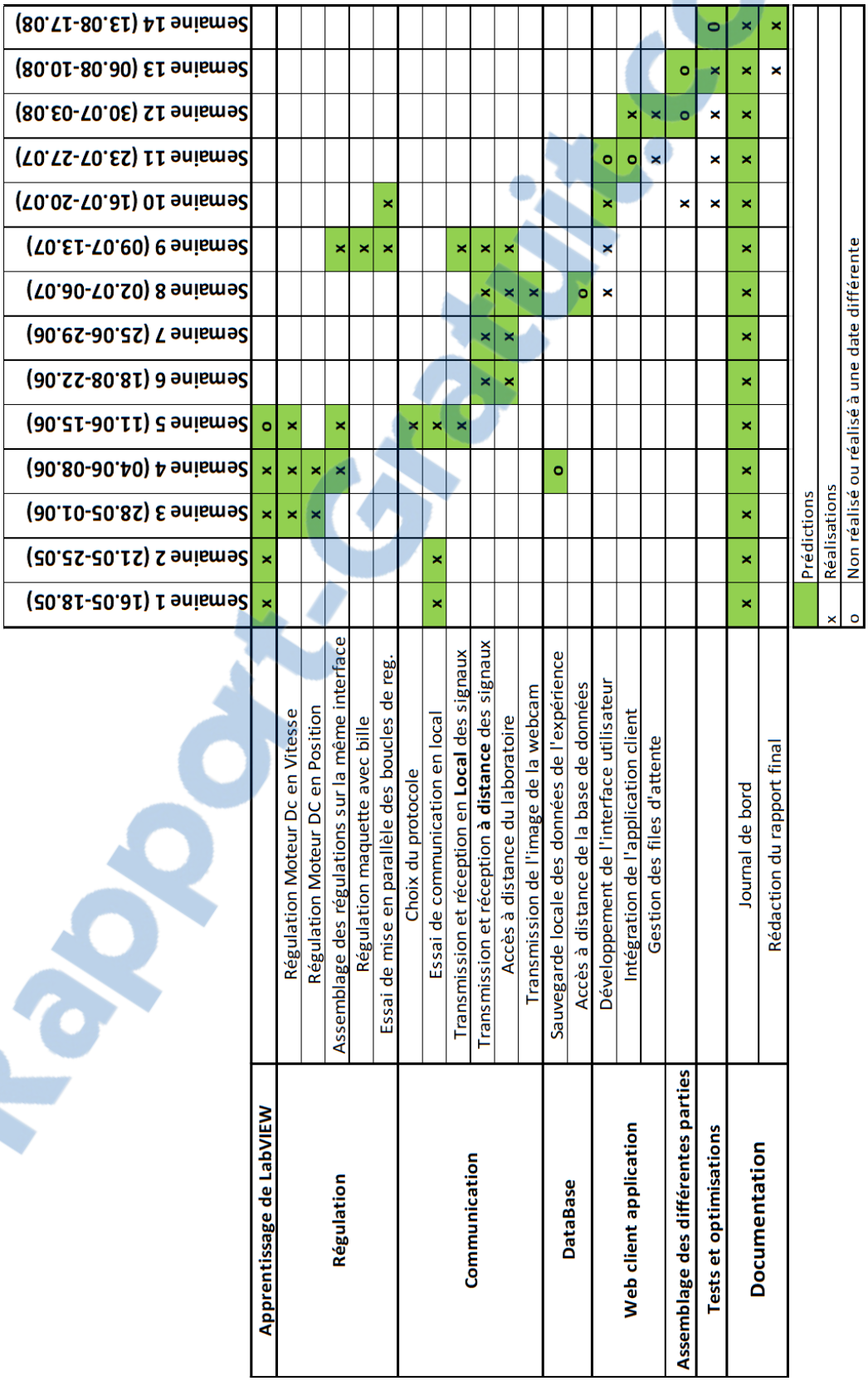
[2] Script Systèmes d'information HES-SO Valais. *Réseaux TCP/IP Architecture.*

[3] Internet Engineering Task Force (IETF). The websocket protocol. `https://tools.ietf.org/html/rfc6455`, December 2011. (Consulté le 8 juillet 2018).

[4] Guy PUJOLLE. Architecture tcp/ip. *Techniques de l'ingénieur Architecture des systèmes et réseaux*, base documentaire : TIB303DUO.(ref. article : h2288), 1997.

12 Annexes

Annexe A Planning



Annexe B Datasheet capteur optique

Optoelektronische Sensoren
Photo electric sensors
Cellules opto-électroniques

FADK 14U4470/S14/IO

IO-Link

Distanz Sensor

Distance Sensor

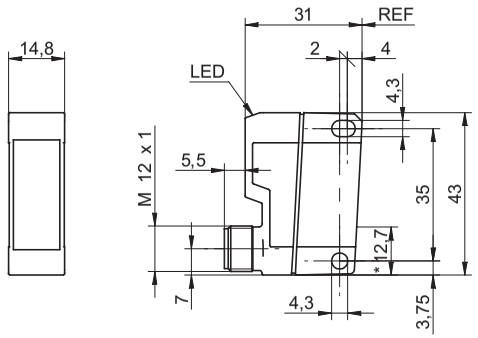
Détecteur de distance

11014499

Baumer

Baumer Electric AG · CH-8501 Frauenfeld
Phone +41 (0)52 728 1122 · Fax +41 (0)52 728 1144

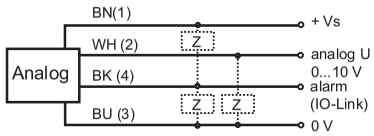
Abmessungen Dimensions Dimensions



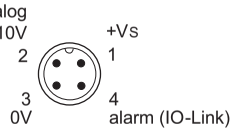
- Alle Masse in mm
- All dimensions in mm
- Toutes les dimensions en mm

* Senderachse
* Emitter axis
* Axe de l'émetteur

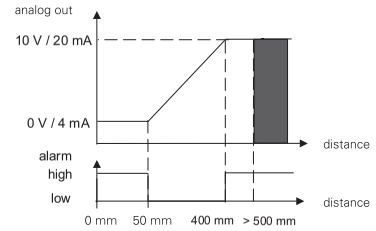
Elektrischer Anschluss Connection diagram Schéma de raccordement



BN = Braun/brown/brun
WH = Weiss/white/blanc
BK = Schwarz/black/noir
BU = Blau/blue/bleu



Ausgang Charakteristik Output characteristic Caractéristique de sortie



- **Vor dem Anschliessen des Sensors die Anlage spannungsfrei schalten.**
- **Disconnect power before connecting the sensor.**
- **Mettre l`installation hors tension avant le raccordement du détecteur.**

Technische Daten	Technical data	Données techniques	
Messbereich	Measuring range	Plage de mesure	50...400 mm
Auflösung (matt weisse Keramik)	Resolution (matt white ceramic)	Résolution (céramique blanche mate)	0,1...1 mm
Linearitätsabweichung (matt weisse Keramik)	Linearity error (matt white ceramic)	Déviation de linéarité (céramique blanche)	± 1,5... ± 4 mm
Lichtquelle	Light source	Source de lumière	pulsed point source LED
Ansprechzeit	Response time	Temps d'activation	< 3 ms
Analogausgang	Analog output	Sortie analogique	0...10 V
Alarmausgang	Alarm output	Sortie d'alarme	push-pull
Betriebsanzeige	Power indicator	Affichage de fonctionnement	green LED
Anzeige Alarm- / Verschm.	Soiled lens indicator	Indication alarme / encrassement	red LED
Betriebsspannungsbereich Vs (UL-Class 2)	Voltage supply range Vs (UL-Class 2)	Plage de tension Vs (UL-Class 2)	14...26 VDC
max. Stromverbrauch	max. supply current	Consommation max.	< 80 mA
Kurzschlussfest	Short circuit protection	Protégé contre courts-circuits	ja / yes / oui
Verpolungsfest	Reverse polarity protection	Protégé contre inversion de polarité	ja / yes / oui *
Arbeitstemperaturbereich	Operating temperature range	Température en service	0...50 °C
Schutzklasse	Protection class	Classe de protection	IP 67
Temperaturdrift	Temperature drift	Dérive en température	< 0,1 % Sde/K

Siehe Betriebsanleitung auf www.baumer.com **See manual on www.baumer.com** **Voir le manuel sur www.baumer.com**

IO-Link

* nur Betriebsspannung / voltage supply only / plage de tension

Technische Änderungen vorbehalten Technical specifications subject to change Sous réserve de modifications techniques

Canada
Baumer Inc.
CA-Burlington, ON L7M 4B9
Phone +1 (1)905 335-8444

China
Baumer (China) Co., Ltd.
CN-201612 Shanghai
Phone +86 (0)21 6768 7095

Denmark
Baumer A/S
DK-8210 Aarhus V
Phone: +45 (0)8931 7611

France
Baumer SAS
FR-74250 Fillinges
Phone +33 (0)450 392 466

Germany
Baumer GmbH
DE-61169 Friedberg
Phone +49 (0)6031 60 07 0

India
Baumer India Private Limited
IN-411058 Pune
Phone +91 20 66292400

Italy
Baumer Italia S.r.l.
IT-20090 Assago, MI
Phone +39 (0)2 45 70 60 65

Singapore
Baumer (Singapore) Pte. Ltd.
SG-339412 Singapore
Phone +65 6396 4131

Sweden
Baumer A/S
SE-56133 Huskvarna
Phone +46 (0)36 13 94 30

Switzerland
Baumer Electric AG
CH-8501 Frauenfeld
Phone +41 (0)52 728 1313

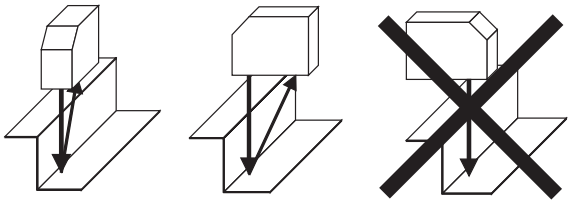
United Kingdom
Baumer Ltd.
GB-Watchfield, Swindon, SN6 8TZ
Phone +44 (0)1793 783 839

USA
Baumer Ltd.
US-Southington, CT 06489
Phone +1 (1)860 621-2121

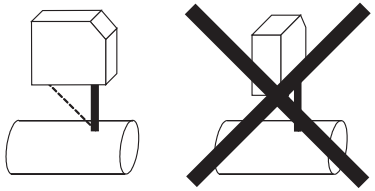
www.baumer.com/worldwide

Montage
Mounting
Montage

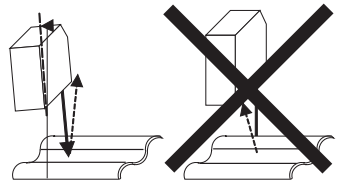
Stufen
Steps
Gradins



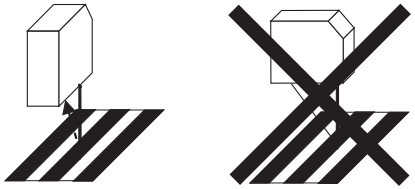
Runde, glänzende Oberflächen
Round glossy surfaces
Surfaces ronds brillantes



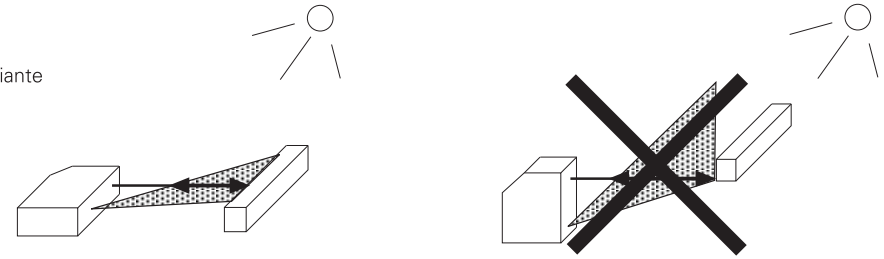
Glänzende Oberflächen
Glossy surfaces
Surfaces brillantes



Unterschiedlich reflektierende Oberflächen
Different reflection of surfaces
Surfaces differemment réfléchissantes



Einwirkung Fremdlicht
Effect of ambient light
Influence lumiére ambiante



Annexe C Datasheet Webcam Logitech C270

Logitech®

HD Webcam C270

HD video calling that's simple.

- **HD 720p video calling on most major IM applications and Logitech Vid™ HD**
- **3-megapixel pics (software enhanced)**
- **Logitech® RightLight™ technology**
- **Logitech® RightSound™ technology**

Logitech® HD Webcam C270. HD video calling that's simple. You'll enjoy an HD 720p video call on most major instant messaging applications and Logitech Vid™ HD-the free, fast and easy way to see your loved ones face to face. Send your family and friends a picture when you don't have time to talk or record a video. Take a 3MP photo (software enhanced) with just one click of a button. Your webcam will adjust to poor lighting conditions automatically to produce the best possible image-thanks to Logitech RightLight™ technology. And background noise doesn't have to spoil your video calls. Logitech RightSound™ technology delivers clear conversations-so you can be heard.

HD 720

CONFIGURATION MINIMALE REQUISE

Windows® XP (SP2 or higher), Windows Vista® or Windows® 7 (32-bit or 64-bit)

Basic requirements:
- 1 GHz
- 512 MB RAM or more
- 200 MB hard drive space
- Internet connection
- USB 1.1 port (2.0 recommended)

For HD 720p video calling and recording:
- 2.4 GHz Intel® Core™2 Duo
- 2 GB RAM
- 200 MB hard drive space
- USB 2.0 port
- 1 Mbps upload speed or higher
- 1280 x 720 screen resolution

Logitech Vid™

CERTIFIED FOR Windows Vista

Designed for Microsoft® Windows® SP2

Compatible with Windows 7

Logitech®

HD Webcam C270

TECHNICAL SPECIFICATIONS

- HD video calling (1280 x 720 pixels) with recommended system
- Video capture: Up to 1280 x 720 pixels
- Photos: Up to 3.0 megapixels (software enhanced)
- Built-in mic with Logitech RightSound™ technology
- Hi-Speed USB 2.0 certified (recommended)
- Universal clip fits laptops, LCD or CRT monitors

Logitech webcam software:
- Logitech Vid™ HD
- Logitech RightLight™ technology
- Video and photo capture

Works with most instant messaging applications

PACKAGE SPECIFICATIONS

	Primary Pack	Master Shipper Carton
Part # WER	960-000582	n/a
Bar code	5099206023758 (EAN-13)	50992060237517 (SCC-14)
Part # CENTRAL	960-000635	n/a
Bar code	5099206023802 (EAN-13)	50992060238019 (SCC-14)
Part # EER	960-000636	n/a
Bar code	5099206023819 (EAN-13)	50992060238118 (SCC-14)
Weight	226.8g	2.147 kg
Length	20.95cm	31.50cm
Width	7.62 cm	31.06cm
Height/depth	15.24 cm	22.40 cm
Volume	2.433dm³	0.02230m³
1 primary pack	1	n/a
1 intermediate pack	0	n/a
1 master shipper carton	8	1
1 pallet EURO	432	54
1 container 20 ft	10080	1260
1 container 40 ft	21280	2660
1 container 40 ft HQ	23408	2926

PACKAGE CONTENT

- Webcam with 5-foot cable
- Headset
- Logitech webcam software with Logitech Vid™ HD
- User documentation

Logitech Vid™

Logitech® **Vid™. The free, fast, and easy way to make video calls with your Logitech webcam.**

www.logitech.com

Annexe D Datasheet cRIO-9067, NI-9201, NI-9263, NI-9411

SPECIFICATIONS

NI cRIO-9067

Embedded Real-Time Controller with Reconfigurable FPGA for C Series Modules

This document lists the specifications for the NI cRIO-9067. The following specifications are typical for the -20 °C to 55 °C operating temperature range unless otherwise noted.

Caution Do not operate the cRIO-9067 in a manner not specified in this document. Product misuse can result in a hazard. You can compromise the safety protection built into the product if the product is damaged in any way. If the product is damaged, return it to NI for repair.

Processor

Type	Xilinx Zynq-7000, XC7Z020 All Programmable SoC
Architecture	ARM Cotex-A9
Speed	667 MHz
Cores	2
Flash reboot endurance[1]	100,000 cycles

Operating System

Note For minimum software support information, visit *ni.com/info* and enter the Info Code `swsupport`.

Supported operating system	NI Linux Real-Time (32-bit)

[1] You can increase the flash reboot endurance value by performing field maintenance on the device. If you expect that your application may exceed the maximum cycle count listed in this document, contact NI support for information about how to increase the reboot endurance value.

NATIONAL INSTRUMENTS

Software requirements	
Application software	
LabVIEW	LabVIEW 2014 or later, LabVIEW Real-Time Module 2014 or later, LabVIEW FPGA Module 2014 or later[2]
Driver software	NI-RIO Device Drivers 14.0 or later

Memory

Nonvolatile memory[3]	1 GB
Volatile memory (DRAM)	512 MB

Network

Network interface	10Base-T, 100Base-T, 1000Base-T Ethernet
Compatibility	IEEE 802.3
Communication rates	10 Mbps, 100 Mbps, 1,000 Mbps auto-negotiated
Maximum cabling distance	100 m/segment

Internal Real-Time Clock

Accuracy	5 ppm

USB Ports

USB device port	
Type	USB 2.0 Hi-Speed, with standard B connector
Maximum data rate	480 Mbps
USB host port	
Type	USB 2.0 Hi-Speed, with standard A connector
Maximum data rate	480 Mbps

[2] LabVIEW FPGA Module is not required when using Scan Interface mode. To program the user-accessible FPGA on the cRIO-9067, LabVIEW FPGA Module is required.

[3] Formatted capactity of nonvolatile memory may be slightly less than this value.

Reconfigurable FPGA

Type	Xilinx Zynq-7000, XC7Z020 All Programmable SoC
Number of logic cells	85,000
Number of flip-flops	106,400
Number of 6-input LUTs	53,200
Number of DSP slices (18 × 25 multipliers)	220
Available block RAM	4480 kbits
Number of DMA channels	16
Number of logical interrupts	32

Battery

Note The battery is not user-replaceable. Refer to the *Battery Replacement and Disposal* section for information about replacing the battery.

Note Battery life may drop dramatically in extreme temperatures.

Typical battery life with power applied to power connector	10 years
Typical battery life in storage at 55 °C	5 years

Power Requirements

Voltage input range	9 VDC to 30 VDC
Reverse-voltage protection	30 VDC maximum
Maximum power input, with four C Series modules	25 W
Maximum power input, without C Series modules	17 W

NI 9201 Specifications

The following specifications are typical for the range -40 °C to 70 °C unless otherwise noted. All voltages are relative to COM unless otherwise noted.

Caution Do not operate the NI 9201 in a manner not specified in this document. Product misuse can result in a hazard. You can compromise the safety protection built into the product if the product is damaged in any way. If the product is damaged, return it to NI for repair.

Input Characteristics

Number of channels	8
ADC resolution	12 bits
Type of ADC	Successive approximation register (SAR)

Table 1. Sample Rate (Aggregate)

Mode	Maximum Sample Rate (R Series Expansion Chassis)	Maximum Sample Rate (All Other Chassis)
Single Channel	475 kS/s	800 kS/s
Scanning	475 kS/s	500 kS/s

Input range	±10 V
Measurement voltage, channel-to-COM (V)	
Minimum	±10.3
Typical	±10.53
Maximum	±10.8
Overvoltage protection, channel-to-COM	±100 V

Table 2. NI 9201 Accuracy (Excludes Noise)

Measurement Conditions		Percent of Reading (Gain Error)	Percent of Range[1] (Offset Error)
Calibrated	Typical (25 °C, ±5 °C)	±0.04%	±0.07%
	Maximum (-40 °C to 70 °C)	±0.25%	±0.25%

[1] Range equals 10.53 V

Table 2. NI 9201 Accuracy (Excludes Noise) (Continued)

Measurement Conditions		Percent of Reading (Gain Error)	Percent of Range[1] (Offset Error)
Uncalibrated[2]	Typical (25 °C, ±5 °C)	±0.26%	±0.46%
	Maximum (-40 °C to 70 °C)	±0.67%	±1.25%

Stability	
Gain drift	±34 ppm/°C
Offset drift	±100 µV/°C
Input bandwidth (-3 dB)	690 kHz min
Input impedance	
Resistance	1 MΩ
Capacitance	5 pF
Input noise, code-centered	
RMS	0.7 LSBrms
Peak-to-peak	5 LSB
No missing codes	12 bits
DNL	-0.9 to 1.5 LSB
INL	±1.5 LSB
Crosstalk, at 10 kHz	-75 dB
Settling time, to 1 LSB	2 µs
MTBF	1,092,512 hours at 25 °C; Bellcore Issue 2, Method 1, Case 3, Limited Part Stress Method

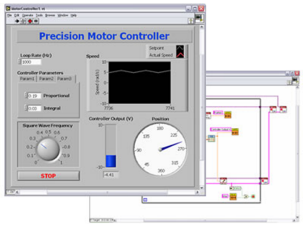
Power Requirements

Power consumption from chassis	
Active mode	1 W maximum
Sleep mode	1 mW maximum
Thermal dissipation (at 70 °C)	
Active mode	1 W maximum
Sleep mode	32 mW maximum

[1] Range equals 10.53 V.

[2] Uncalibrated accuracy refers to the accuracy achieved when acquiring in raw or unscaled modes where the calibration constants stored in the module are not applied to the data.

NI LabVIEW Real-Time Module

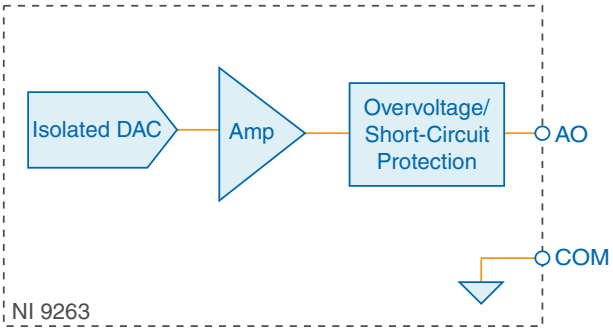


- Design deterministic real-time applications with LabVIEW graphical programming
- Download to dedicated NI or third-party hardware for reliable execution and a wide selection of I/O
- Take advantage of built-in PID control, signal processing, and analysis functions
- Automatically take advantage of multicore CPUs or set processor affinity manually
- Take advantage of real-time OS, development and debugging support, and board support
- Purchase individually or as part of a LabVIEW suite

Circuitry

Each channel has a digital-to-analog converter (DAC) that produces a voltage signal. Each channel also has overvoltage and short-circuit protection.

Figure 1. Output Circuitry for One Channel of the NI 9263



NI 9263 Specifications

The following specifications are typical for the range -40 °C to 70 °C unless otherwise noted. All voltages are relative to COM unless otherwise noted.

Caution Do not operate the NI 9263 in a manner not specified in this document. Product misuse can result in a hazard. You can compromise the safety protection built into the product if the product is damaged in any way. If the product is damaged, return it to NI for repair.

Output Characteristics

Number of channels	4 analog output channels
DAC resolution	16 bits

Type of DAC	String
Power-on output state	Channels off
Startup voltage[1]	0 V
Power-down voltage[2]	0 V
Output voltage range	
Nominal	±10 V
Minimum	±10.4 V
Typical	±10.7 V
Maximum	±11 V
Current drive	±1 mA per channel maximum
Output impedance	2 Ω

Table 1. Accuracy

Measurement Conditions		Percent of Reading (Gain Error)	Percent of Range[3] (Offset Error)
Calibrated	Maximum (-40 °C to 70 °C)	0.35%	0.75%
	Typical (25 °C, ±5 °C)	0.03%	0.1%
Uncalibrated[4]	Maximum (-40 °C to 70 °C)	2.2%	1.7%
	Typical (25 °C, ±5 °C)	0.3%	0.25%

Stability	
Gain drift	11 ppm/°C
Offset drift	110 µV/°C
Protection	
Overvoltage	±30 V
Short-circuit	Indefinitely

1 When the module powers on, a glitch occurs for 20 µs peaking at -1.5 V.
2 The power-down voltage peaks at 1.8 V before exponentially discharging to 0 V in 100 µs. You can add a 10 kΩ load to reduce the peak voltage.
3 Range equals ±10.7 V
4 Uncalibrated accuracy refers to the accuracy achieved when acquiring in raw or unscaled modes where the calibration constants stored in the module are not applied to the data.

Table 2. Update Time

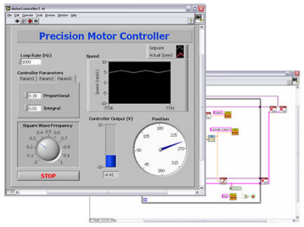
Number of Channels	Update Time for All Other Chassis	Update Time for NI cRIO-9151 R Series Expansion Chassis
1	3 µs min	3.5 µs min
2	5 µs min	6.5 µs min
3	7.5 µs min	9 µs min
4	9.5 µs min	12 µs min

Noise	
Updating at 100 kS/s	600 µVrms
Not updating	260 µVrms
Slew rate	4 V/µs
Crosstalk	76 dB
Settling time (100 pF load, to 1 LSB)	
Full-scale step	20 µs
1 V step	13 µs
0.1 V step	10 µs
Capacitive drive	1,500 pF minimum
Monotonicity	16 bits
DNL	±1 LSB maximum
INL (endpoint)	±12 LSB maximum
MTBF	1,732,619 hours at 25 °C; Bellcore Issue 2, Method 1, Case 3, Limited Part Stress Method

Power Requirements

Power consumption from chassis	
Active mode (at -40 °C)	500 mW maximum
Sleep mode	25 µW maximum
Thermal dissipation (at 70 °C)	
Active mode	750 mW maximum
Sleep mode	25 µW maximum

NI LabVIEW Real-Time Module

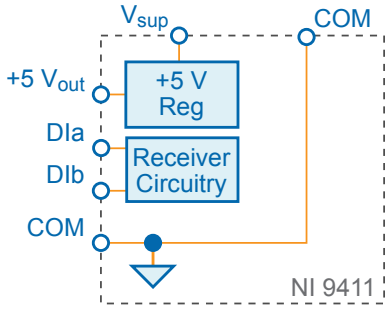


- Design deterministic real-time applications with LabVIEW graphical programming
- Download to dedicated NI or third-party hardware for reliable execution and a wide selection of I/O
- Take advantage of built-in PID control, signal processing, and analysis functions
- Automatically take advantage of multicore CPUs or set processor affinity manually
- Take advantage of real-time OS, development and debugging support, and board support
- Purchase individually or as part of a LabVIEW suite

Input Circuitry

The NI 9411 channels share a common ground isolated from other modules in the system.

Figure 1. NI 9411 Input Circuitry



NI 9411 Specifications

The following specifications are typical for the range -40 °C to 70 °C unless otherwise noted. All voltages are relative to COM unless otherwise noted.

Caution Do not operate the NI 9411 in a manner not specified in this document. Product misuse can result in a hazard. You can compromise the safety protection built into the product if the product is damaged in any way. If the product is damaged, return it to NI for repair.

Input Characteristics

Number of channels	6 digital input channels
Input type	Differential or single-ended

Digital logic levels	
Differential (DIa and DIb)	
Input high range	300 mV to 24 V
Input low range	-300 mV to -24 V
Common-mode voltage	-7 V to 12 V
Single-ended	
Input high range	2 V to 24 V
Input low range	0 V to 0.8 V
Input current	
At 5 V	±1 mA per channel
At 24 V	±4 mA per channel
Input impedance	8.4 kΩ
I/O protection	
Input voltage (channel-to-COM)	30 V maximum
Input current	±4 mA, internally limited
Input delay time	500 ns maximum
MTBF	800,319 hours at 25 °C; Bellcore Issue 2, Method 1, Case 3, Limited Part Stress Method

Power Requirements

Power consumption from chassis	
Active mode	340 mW maximum
Sleep mode	1.1 mW maximum
Thermal dissipation (at 70 °C)	
Active mode	1.4 W maximum
Sleep mode	1.1 W maximum

External Power Supply

Input voltage range (Vsup)	5 VDC to 30 VDC maximum

Annexe E Création d'un exécutable LabVIEW

Créer un exécutable autonome sous LabVIEW

Date de publication: avr. 19, 2012

Introduction

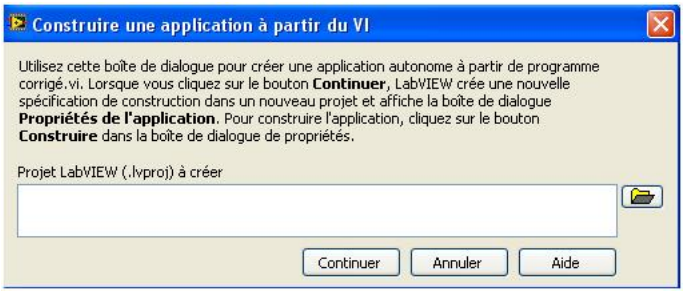
Ce document propose de détailler la création d'un exécutable avec LabVIEW 2010. Pour créer un exécutable autonome, vous devez disposer de la version professionnelle de LabVIEW (http://www.ni.com/labview/f/) (Professional Development System) contenant l'Application Builder.
Vous pouvez vérifier la version présente sur l'ordinateur via le Gestionnaire de licences NI.

L'Application Builder va vous permettre de créer des applications autonomes pour fournir des versions exécutables de VIs à d'autres utilisateurs. Les applications sont utiles lorsque vous voulez que les utilisateurs puissent exécuter des VIs sans avoir à installer le système de développement LabVIEW. Néanmoins les exécutables créés exigent la présence du moteur d'exécution LabVIEW qui sera distribué par l'installeur ou téléchargeable sur http://www.ni.com/downloads/ (http://www.ni.com/downloads/).
Sur Windows, les applications ont l'extension « .exe » et sur Mac OS « .app ». En contre partie, l'utilisateur de l'exécutable n'aura pas accès au code (diagramme) du programme. Il aura devant lui un instrument virtuel exécutant les tâches configurées auparavant, lors de l'écriture du VI. Un exécutable contient l'ensemble des sous-VIs constituant le programme initial.
Avant de pouvoir créer une application autonome avec vos VIs, vous devez d'abord préparer vos fichiers pour la distribution. Reportez-vous à la section Préparation des fichiers de la Liste de construction d'applications de l'Aide LabVIEW.

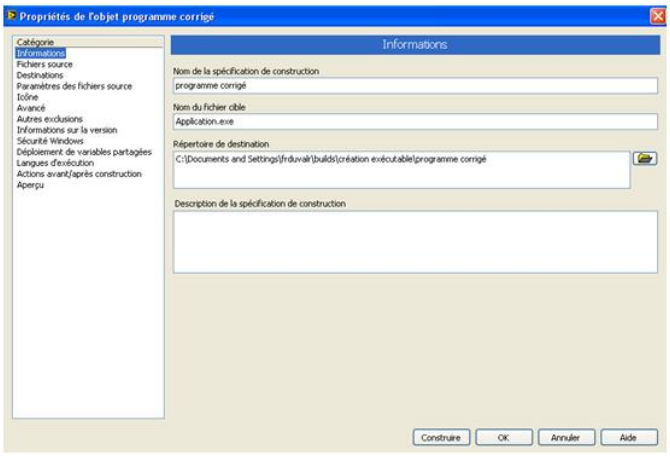
Création du .exe

Si vous avez créé votre dossier sans projet, ouvrez votre VI principal. Cliquez sur **Outils** >> **Construire l'application (EXE) à partir d'un VI…**

Une fenêtre vous demandant un chemin où créer le projet apparaît. Un chemin par défaut est déjà présent.

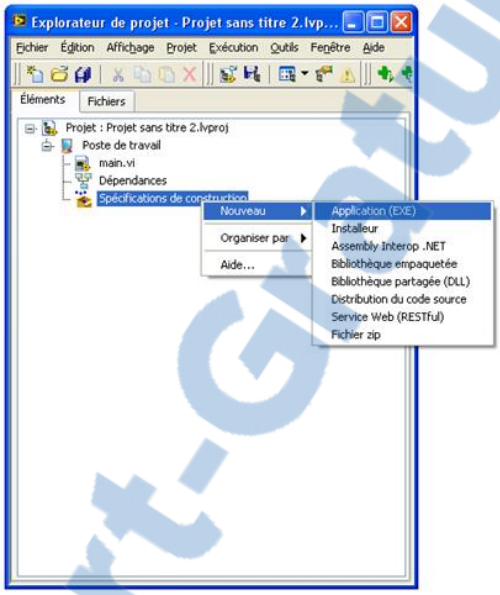


Appuyez ensuite sur **continuer**. Une boîte de dialogue avec de multiples onglets apparaît :



1.

Si vous aviez déjà développé votre application sous forme de projet, pour arriver à la même fenêtre, il vous suffit d'effectuer un clic droit sur **Spécifications de construction** et choisir Application (EXE).



La fonction des différents onglets de la partie catégorie va maintenant être détaillée.

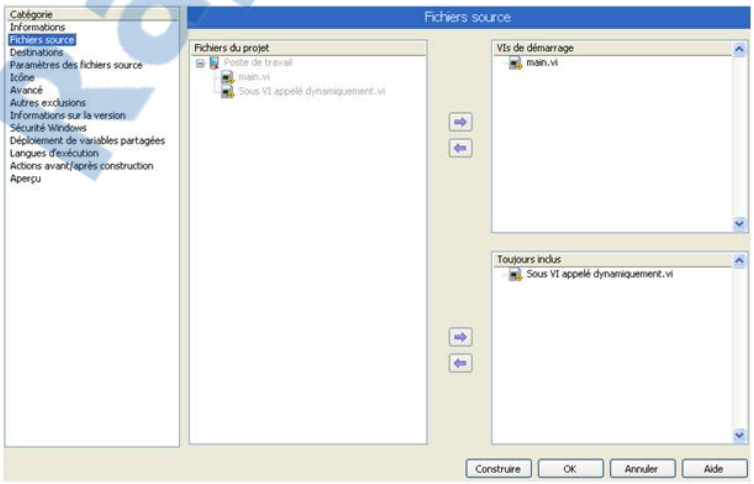
Informations

Utilisez cette fenêtre pour renommer votre exécutable et sélectionner le dossier où sera construit l'exécutable.

Fichiers source

Dans cet écran, vous devez placer uniquement le VI principal de plus haut niveau dans la partie VI de démarrage. LabVIEW inclut automatiquement tous les sous-VIs utilisés dans le VI principal.

La seule exception est l'ajout des VIs appelés dynamiquement par VI server. Si votre application comporte de tels VIs, ajoutez-les dans la partie Toujours inclus.



Destinations

Cette fenêtre vous permet de spécifier où seront placés les différents fichiers créés lors de la construction de l'exécutable (entre autres le fichier.exe et les fichiers de support).

Paramètres des fichiers sources

Vous pourrez y personnaliser les propriétés du VI sélectionné dans l'arborescence **Fichiers du projet** et notamment l'aspect de sa face-avant à l'exécution. Il vous sera aussi possible de gérer les mots de passe que vous auriez éventuellement appliqués aux VIs.

Icône

Utilisez cette fenêtre afin de créer une icône personnalisée pour votre exécutable.

Avancé

Utilisez cette page pour changer les paramètres avancés de votre exécutable. Cette partie est notamment importante si des ActiveX ou graphes 3D sont présents au sein de votre application. Reportez-vous à la partie Page avancée de l'Aide LabVIEW pour plus de détails.

Autres exclusions

Utilisez cette page de la boîte de dialogue pour configurer des paramètres permettant de supprimer ou de déconnecter les définitions de type, les instances de VIs polymorphes inutilisées et les membres inutilisés des bibliothèques de projet. Utilisez ces paramètres pour réduire la taille de l'application autonome.

Informations sur la version

Cette fenêtre vous permet de spécifier la version de votre exécutable et d'ajouter diverses informations liées au développement de celui-ci.

Sécurité Windows

Fenêtre permettant de configurer les paramètres de sécurité des ordinateurs qui fonctionnent sous Windows pour votre application autonome. Reportez-vous au Centre d'aide et de support Windows(cliquez sur Démarrer»Aide et support) pour obtenir des informations complémentaires sur la configuration des paramètres de sécurité.

Déploiement de variables partagées

Fenêtre où vous pourrez spécifier quelle bibliothèque dépendante contenant des variables partagées doit être déployée lors de l'exécution.

Langues d'exécution

Vous pourrez y sélectionner la langue utilisée par votre exécutable. Les préférences de langue s'appliquent aux aspects de l'application autonome affectés par le moteur d'exécution de LabVIEW comme par exemple, les boîtes de dialogue et les menus. Ces éléments apparaissent dans la langue par défaut que vous sélectionnez.

Actions avant/après construction

Vous permet de tester votre exécutable avant et/ou après construction afin de savoir si le résultat final est satisfaisant.

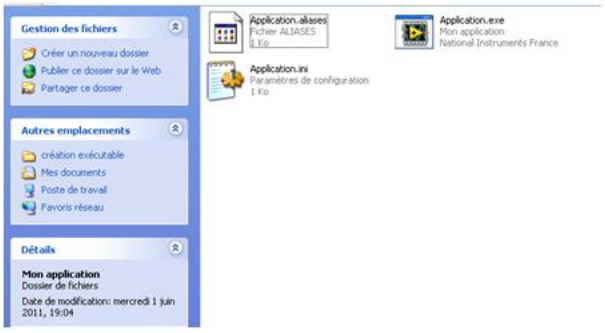
Aperçu

Fenêtre vous permettant d'avoir un aperçu du dossier final créé et contenant votre exécutable et les divers documents.

Construction de l'exécutable

Une fois les divers paramètres réglés selon vos préférences, cliquez sur Construire pour lancer la création de l'exécutable. Une fenêtre vous permet de visualiser la progression de la construction :

Une fois l'exécutable créé, vous pouvez explorer le dossier créé :



Pour la création de l'installeur (nécessaire pour ajouter les moteurs d'exécution et donc pour déployer l'exécutable sur des ordinateurs ne disposant pas de LabVIEW) ainsi que pour un résumé sur la méthode de création d'exécutables, suivez le lien suivant :

Distributing Applications with the LabVIEW Application Builder (http://zone.ni.com/devzone/cda/tut/p/id/3303#toc4)

Annexe F Création d'un installeur LabVIEW

Building an Installer (Windows)

Table of Contents (/reference/en-XX/help/371361H-01)

LabVIEW 2011 Help

Edition Date: June 2011

Part Number: 371361H-01

»View Product Info (http://sine.ni.com/nips/cds

Download Help (Winc (http://www.ni.com/pdf/

You can use the Application Builder to build an installer for files in a LabVIEW project or for a stand-alone application (../building_a_stand_alone_app/), shared library (../building_a_dll/), or source distribution (../build_source_distrib/) you created with a build specification. Before you create a build specification or start the build process, review the caveats and recommendations (../caveats_install/) for LabVIEW installer builds.

Note The LabVIEW Professional Development System includes the Application Builder. If you use the LabVIEW Base Package or Full Development System, you can purchase the Application Builder separately by visiting the National Instruments Web site.

Complete the following steps to build an installer. Refer to the NI Developer Zone for a step-by-step tutorial on creating a sample installer.

Configuring the Build Specification

1. Open the project you want to use to build the installer. You must have a project open and saved to configure an installer build specification.
2. Expand **My Computer**. Right-click **Build Specifications** and select **New»Installer** from the shortcut menu to display the Installer Properties (../../lvdialog/installer_tab_windows/) dialog box. If you previously hid **Build Specifications** in the **Project Explorer** window, you must display the item (../showing_hiding_nodes/) again to access it.
3. Complete the following items on the Product Information (../../lvdialog/product_info_page/) page of the **Installer Properties** dialog box.
    1. Enter a name for the installer build specification in the **Build specification name** text box. This name appears under **Build Specifications** in the **Project Explorer** window. The name must be unique among other build specification names in the project.
    2. Enter a name for the product associated with the installer in the **Product Name** text box. This name corresponds to the `[ProductName]` Microsoft Installer (MSI) property. Refer to the Microsoft Web site for more information about Microsoft Installer (MSI) properties.
    3. Enter the location for the installer in the **Installer destination** text box. You can use the **Browse** button to navigate to and select a location.
4. From the Destinations (../../lvdialog/destinations_install_page/) page, configure the destination directory of the installer. Use the options below the **Destination View** tree to add a new destination directory, add a folder to an existing destination directory, add a new absolute path, and remove a new directory.
5. From the Source Files (../../lvdialog/files_install_page/) page, select the files or builds you want to include in the installer from the **Project Files View** tree. Use the **Add** button to add the selected files and builds in the **Project Files View** to the folders in the **Destination View** tree. Use the **Remove** button to remove the selected files and builds from the **Destination View** tree.
6. From the Source File Settings (../../lvdialog/file_set_install_page/) page, place checkmarks in the checkboxes in the **File and Folder Attributes** section to configure the file attributes at installation. If the installer build includes a stand-alone application in which you enabled the ActiveX server, place a checkmark in the **Register COM** checkbox for the application file. To remove administrator access requirements from deployed files and folders, place a checkmark in the **Unlock** checkbox to unlock the selected file or folder.

Note To set file attributes for dependent files, you must add them manually to the project. In the **Project Explorer** window, right-click **My Computer** and select **Add»File** from the shortcut menu. Navigate to and select the dependent file. Refer to steps 5 and 6 in the previous procedure to add the file to the **Destination View** tree and configure file attributes.

7. From the Version Information (../../lvdialog/version_install_page/) page, enter the version, dialog box settings, and company information of the installer. You also can generate a new upgrade code so that when you duplicate an installer (../build_installer/), the new installer does not replace the previous version.

Customizing Advanced Installer Options

1. From the Shortcuts (../../lvdialog/shortcuts_page/) page, configure any shortcuts you want to create to files in the installer.
2. From the Additional Installers (../../lvdialog/addl_install_info_db/) page, add installers for any National Instruments products or drivers (../add_installers_to_build/) to include in the installer build. For example, if you build an installer for an application or shared library that uses the Storage/DataPlugin VIs (../../glang/storage_vis/), you must add the NI USI installer.
3. From the Dialog Information (../../lvdialog/dialoginfo/) page, set the user interface text and dialog box graphics that appear when you run the installer.
4. From the Registry (../../lvdialog/installer_tab_windows/) page, create registry keys and values for the installer.
5. From the Hardware Configuration (../../lvdialog/hardconfig/) page, specify the hardware configuration information to include in the installer build.

Note Include the LabVIEW Run-Time Engine (../../lvconcepts/using_the_lv_run_time_eng/) if you are distributing

applications and shared libraries to computers on which LabVIEW is not installed.

6. From the Windows Security (../../lvdialog/windows_security_install_page/) page, configure the digital signature for `setup.exe` when the installer runs.
7. From the Advanced (../../lvdialog/adv_installer_settings_db/) page, configure advanced settings for the installer.

Building the Installer

1. Click the **OK** button to update the build specification settings in the project and close the dialog box. The build specification name appears in the project under **Build Specifications**.

Note When you update the build specification settings, the settings become part of the project. However, you must save the project if you want to save the settings.

2. Right-click the installer build specification name and select **Build** from the shortcut menu. You can find the resulting installer files in the directory specified in the **Installer destination** text box on the **Product Information** page of the build specification.

You can duplicate build specifications (../../lvconcepts/building_standaloneapps/). Right-click the build specification item you want to duplicate and select **Duplicate** from the shortcut menu to create a copy of the build specification item.

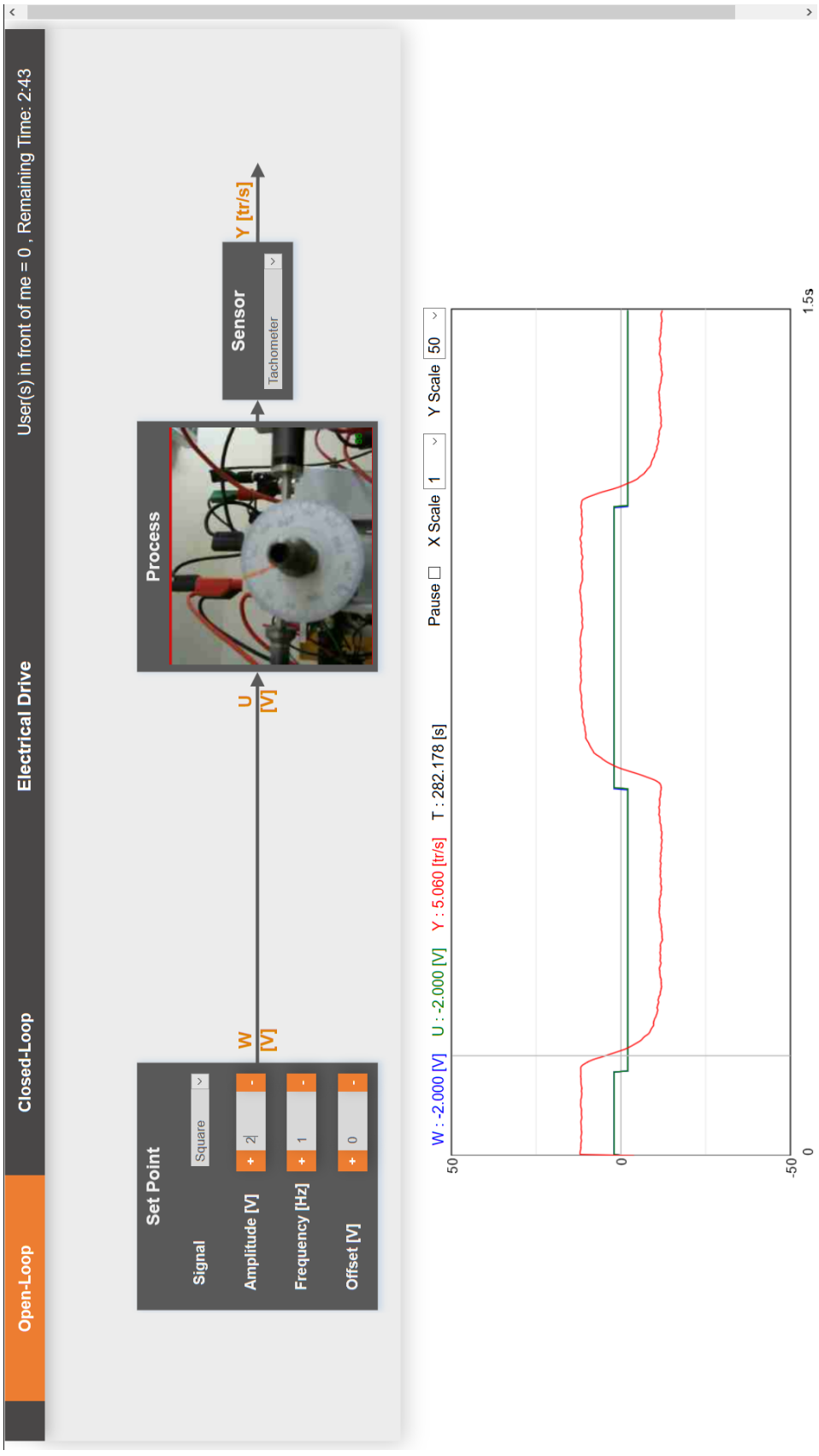
If you rebuild a given specification, LabVIEW overwrites the existing files from the previous build that are part of the current build.

You also can build an installer programmatically from a build specification using the Build (../../glang/build_vi/) VI.

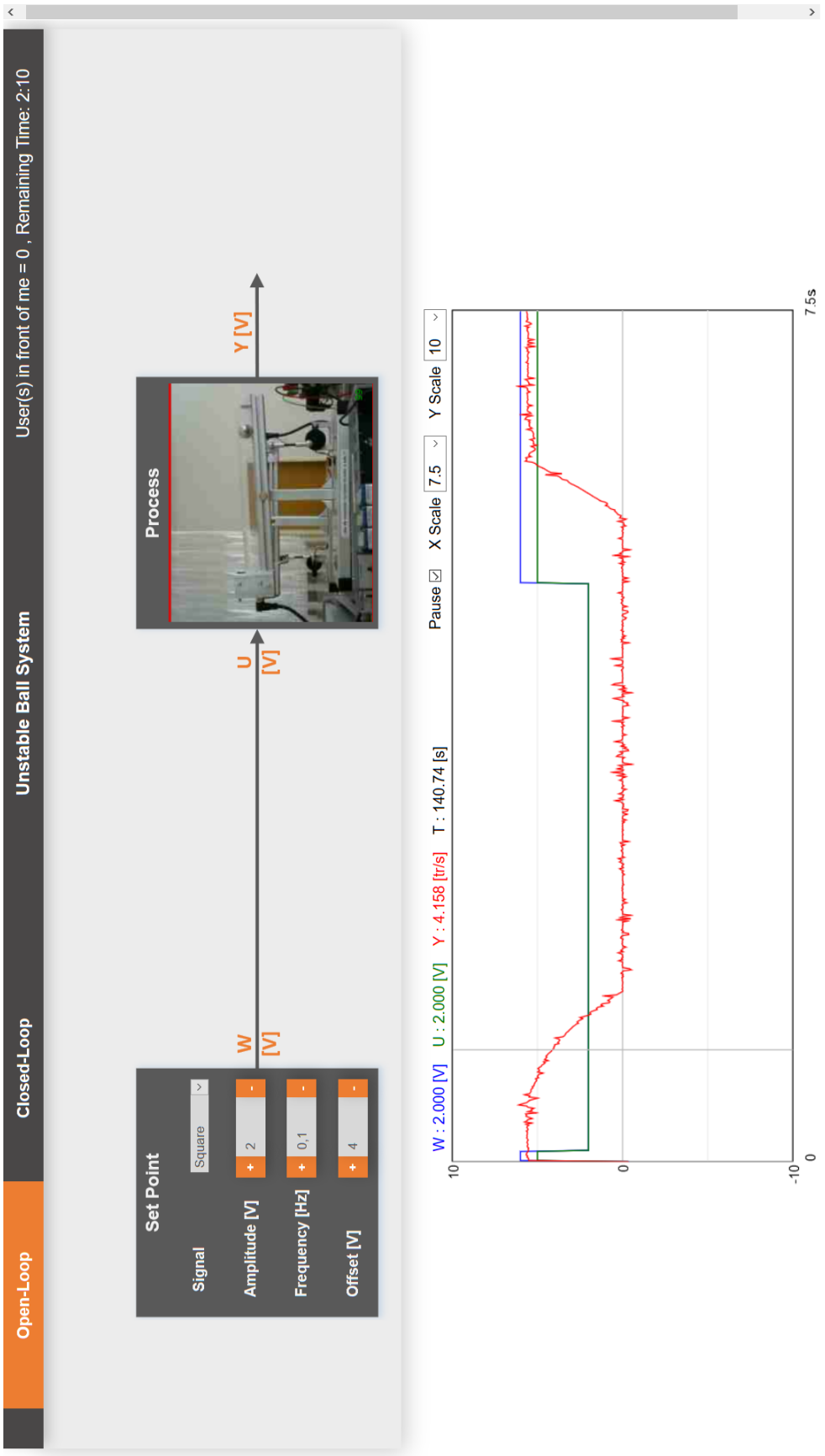
Note You do not need additional license files when using LabVIEW distribution components with LabVIEW-built applications.

(../../lvconcepts/building_and_distributing_applications/) *Building and Distributing Applications* Home (../../lvconcepts/building_and_distributing_applications/)

Annexe G Entraînement Electrique Open-Loop



Annexe H Système instable à bille Open-Loop



Annexe I Système instable à bille Closed-Loop