Plateforme d'objets connectés évolutive pour l'automatisation et la robotisation

par

Félix LAVOIE-ROBERT

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE
AVEC MÉMOIRE EN GÉNIE DE LA PRODUCTION AUTOMATISÉE
M. Sc. A.

MONTRÉAL, LE 14 JUIN 2023

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Tony Wong, directeur de mémoire
Département de génie des systèmes à l'École de technologie supérieure

M. Mustapha Ouhimmou, président du jury
Département de génie des systèmes à l'École de technologie supérieure

M. Julio Montecinons, membre du jury
Département de génie des systèmes à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 15 MAI 2023

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

La conclusion de ma maîtrise et de ce mémoire n'aura pas été sans difficulté durant ces trois dernières années. Au cours de cette période, j'ai eu à faire face à plusieurs épreuves, bien au-delà de ce que je pensais entreprendre initialement.

J'aimerais remercier ma future femme et meilleure amie, Isabel, de m'avoir supporté au cours des dernières années. J'ai hâte de passer le reste de ma vie avec toi. Par le fait même, merci à ma famille (Claude, Simon, Étienne, Marie-Pier, Emma et Jacob) ainsi que ma belle-famille (Marc, Marie-Claude, Jean-Pierre, Laurent et Rosalie).

Il aurait été impossible pour moi d'achever cette maîtrise sans l'aide de mes proches. Cependant, c'est sans aucun doute Tony Wong qui a su m'inspirer à en arriver jusqu'ici. Chaque rencontre m'ouvrait les yeux et m'a permis de ne pas oublier d'éprouver du plaisir à faire ce que je fais. Il va sans dire que cette maîtrise n'aurait probablement jamais été entamée sans l'aide et le financement de Mitacs. En me permettant d'aller faire ma maîtrise, Mitacs a su remplir une mission qui me tient à cœur, celle de l'accès à l'éducation.

J'aimerais sincèrement remercier Claude Beaulieu ainsi que l'ensemble de Sycodal de m'avoir permis de m'épanouir à leur côté. J'aimerais également remercier tous mes collègues qui m'ont côtoyé au courant de ces années. Khaled, Pierre, Samuel et Guillaume, vous faites partie de mes modèles. J'aimerais également remercier Maxime, Thomas, Houssem et Jonathan qui m'ont supporté au travers de cette période de rédaction.

Merci à Olivier, Thierry, Philippe, Benjamin et Antoine et vos douces de faire partie de ma vie. Un petit merci à Couscous et Sarrasin, parce que ce sont les meilleurs chats du monde.

Enfin, j’aimerais dédier ce mémoire à ma mère, Francine. Si j’ai eu la force d’écrire durant cette année, c’est pour voir ta petite face de fierté quand tout sera fini. Tu es d’une force incroyable et une source d’inspiration quotidienne pour moi.

Plateforme d'objets connectés évolutive pour l'automatisation et la robotisation

Félix LAVOIE-ROBERT

RÉSUMÉ

Un besoin critique de main-d'œuvre en Amérique du Nord crée une forte demande en automatisation de procédé. Certaines tâches demandent des prises de décisions et seront très difficiles à automatiser conventionnellement, donc des robots autonomes sont donc proposés comme solutions à ces problèmes. À partir de cette proposition, on suggère une solution de gestion de communication entre un robot autonome et son environnement basé sur les principes de l'internet des objets.

Dans ce mémoire, nous proposons l'utilisation de différentes plateformes de développement, ROS et Node-Red, combinées à l'utilisation de protocoles de communication tels que MQTT et de réseaux de terrain (Fieldbus) pour servir d'infrastructure de communication. Cette passerelle a été conçue avec l'idée d'être utilisable dans une variété de contextes, autant avec des systèmes industriels conventionnels qu'avec des technologies de dernier cri.

Afin de tester cette passerelle de communication, un protocole expérimental a été mené afin d'y obtenir le débit et la latence du système. Enfin, ces résultats ont été analysés avec une méthode statistique non paramétrique. La conclusion de cette expérience est que le système est en mesure de communiquer en temps réel efficacement.

Globalement, une passerelle de communication utilisant les principes de l'internet des objets a été développée afin d'être utilisée par des robots autonomes. Celle-ci devra toutefois être mise à jour dans un futur rapproché en raison de la fin du support à long terme de ROS.

Mot-clés : IoT, Gateway, ROS, MQTT, Node-Red

Evolving platform of connected objects for automation and robotization

Félix LAVOIE-ROBERT

ABSTRACT

A critical labor shortage in North America is creating a strong demand for process automation. Some tasks require decision-making and will be very difficult to automate conventionally, so autonomous robots are proposed as a solution to these problems. Based on this proposal, a solution for managing communication between an autonomous robot and its environment based on the internet of things is suggested.

In this thesis, the use of different development platforms, ROS and Node-Red, combined with the use of communication protocols, MQTT and fieldbus networks, is proposed to serve as a communication infrastructure. This gateway was designed with the idea of being usable in a variety of contexts, both with conventional industrial systems and with cutting-edge technologies.

To test this communication gateway, an experimental protocol was established and followed to obtain the system's throughput and latency. Finally, these results were analyzed using a non-parametric statistical method. The conclusion of this experiment is that the system can communicate efficiently in real-time.

Overall, a communication gateway using the principles of the Internet of Things has been created to be used by autonomous robots. However, it will need to be updated in the near future due to the end of long-term support for ROS.

Keywords: IoT, Gateway, ROS, MQTT, Node-Red

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
IoT	Internet of Things
IIOT	Industrial Internet of Things
JSON	Javascript Object Notation
LCM	Lightweight Communication and Marshalling
LoRa	Long Range
MQTT	Message Queueing Telemetry Transport
MRPT	Mobile Robot Programming Toolkit
OSI	Open System Interconnection
OPC	Open Platform Communication
OPC UA	Open Platform Communication Unified Architecture
Package	Paquet ROS
ROS	Robot Operating System
RTL	Round Trip Latency
TCP/IP	Transport Control Protocol/Internet Protocol
Topic	Sujet
XML	Extensible Markup Language

LISTE DES SYMBOLES ET UNITÉS DE MESURE

Hz	Hertz
msgs/s	Messages par seconde
ms	Milliseconde
s	Seconde

INTRODUCTION

0.1 Contexte et motivation

Les robots autonomes sont conçus pour opérer dans un espace commun avec des humains. L'idée de la cohabitation robot-humain œuvrant conjointement à la réalisation des tâches remonte vers la fin des années 90 (Colgate et al., 1996). C'est au milieu des années 2000 que cette idéation a pris de l'ampleur. C'est donc une technologie relativement récente qui est en pleine émergence. En parallèle à cette innovation dans le domaine de la robotique, la pénurie de la main-d'œuvre qualifiée a été observée dans bien des pays industrialisés. Déjà en 2006, le vieillissement des baby-boomers et la baisse du taux de natalité font craindre un déséquilibre entre l'offre et la demande dans le secteur de la santé (Freeman, 2006). Aujourd'hui au Québec, ce déséquilibre est encore plus profond et les solutions telles l'immigration, la diversification de recrutement et la rétention des employés ne font que repousser le problème plus loin sans régler le problème à la racine.

Ce projet de recherche a été établi à partir d'une proposition selon laquelle les robots autonomes pourront pallier le manque de main-d'œuvre. Cette proposition consiste à faire intervenir un robot dans le volet de l'aide technique des personnes en perte de mobilité dans une résidence privée. Dans le contexte de la conception d'un robot autonome, la communication entre le robot et les dispositifs de l'environnement est un élément critique à considérer. Ce projet de recherche consiste justement à contribuer à la gestion de la communication entre le robot autonome et son environnement selon l'approche de l'internet industriel des objets (IIoT).

La communication, dans le contexte de la robotisation, implique un nombre de dispositifs produits par différents fabricants. Les échanges de données peuvent être réalisés de manières différentes selon les besoins et le type d'application. Le résultat est une prolifération de méthodes de communication et la comptabilité des méthodes de communication n'est pas

toujours assurée. Pour résoudre ce problème de compatibilité, on doit recourir au modèle de base de la communication informatique. La Figure 0.1 présente le modèle en couche de l'organisation Open System Interconnection (OSI).

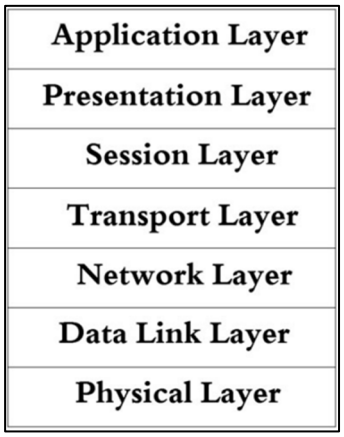

Figure 0.1 Modèle OSI
Tirée de M.Alani (2014)

Ce modèle généraliste est divisé en sept couches de communication standardisées. Commençant du bas, la première couche, la couche physique, représente le module physique de l'émission de donnée. Pensons par exemple à un câble optique ou un câble réseau. La seconde couche représente le lien entre les données. Par exemple, un câble réseau pour communiquer par Ethernet, mais pourrait également être un modèle de communication série comme le USB (*Universal Serial Bus*). La troisième couche est celle du réseau. Cette couche identifiera numériquement chaque dispositif de notre système, avec une adresse IP (*Internet Protocol*) par exemple.

La quatrième couche représente la couche de transport. Elle segmente l'information des couches 2 et 3, puis la transmet aux différents dispositifs. Le protocole de contrôle de transmission (TCP) est généralement utilisé ici. La cinquième couche est celle de la session. Le concept d'une session dans le modèle représente l'instance de la connexion de données entre deux dispositifs parmi tous les dispositifs sur le réseau. La sixième couche est celle de la présentation des données. Par exemple, lors d'une requête de données à un dispositif, celui-ci

peut soumettre ces données dans un format standard tels XML (*Extensible Markup Language*), JSON (*Javascript Object Notation*) ou bien CSV (*Comma Separated Values*). Enfin, la septième couche est celle de l'application. C'est à cet endroit qu'on détermine comment l'utilisateur peut faire appel à son réseau. Par exemple, lorsqu'un réseau est mis en place, l'utilisateur peut utiliser des commandes http (*Hypertext Transmission Protocole*) pour réaliser ses opérations.

Dans les applications industrielles, tous ces concepts sont touchés par les différents contextes de communication. Étant donné la vaste demande et la grande diversité de dispositifs intégrables dans le domaine industriel, la réseautique industrielle s'est développée et des protocoles standards uniques à l'industrie ont été créés. Les choix de protocoles industriels peuvent grandement varier selon différents critères. En effet, malgré qu'ils occupent souvent la même fonction dans un système, les couches 5,6 et 7 du modèle OSI sont lourdement touchées par les variations des protocoles. Les performances varieront d'un protocole à l'autre et certains fabricants favoriseront l'utilisation de leur protocole propriétaire par rapport à ceux de leur compétition. La facilité d'intégration est souvent un critère important de sélection des composants et dispositifs. Habituellement, un réseau industriel contient un ou plusieurs contrôleurs configurables à partir de logiciel propriétaire. Ces contrôleurs auront alors d'autres dispositifs qui agiront comme esclaves et seront utilisés par le maître. La Figure 0.2 est un exemple d'architecture de communication industrielle utilisant le protocole Modbus TCP :

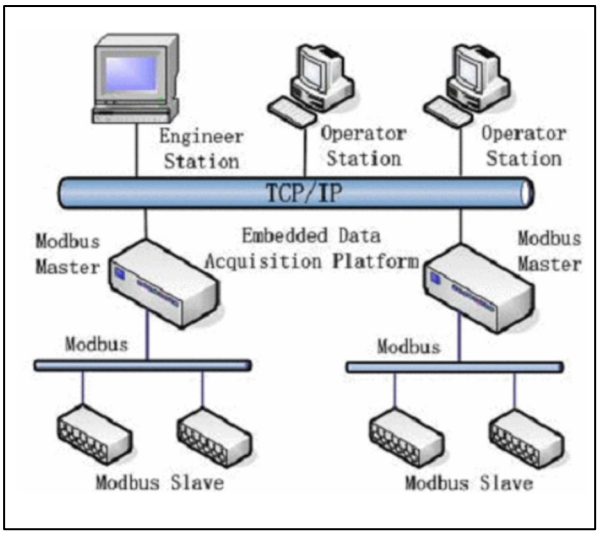

Figure 0.2 Réseau industriel de communication
Tirée de Peng et al. (2008)

Normalement, ces réseaux sont fermés et n'ont aucune connexion externe outre le réseau de la machine. Certains systèmes spécialisés pour superviser et acquérir des données ont été développés pour obtenir des points de vue à haut niveau des machines, des systèmes de contrôle et d'acquisition de données en temps réel (SCADA). Ces systèmes sont utilisés depuis des années et ont été éprouvés au cours du 21e siècle (Ahmed et al.,2015). Ils sont très performants et facilement configurables dans leur système, ce qui les rend très communs dans le domaine. Cependant, ils sont aussi dans le développement fermé et ne sont pas très réceptifs à l'idée de la source ouverte.

Si certains systèmes sont de conception datée et plutôt dépassé, ils sont encore très communs dans l'industrie de production au Québec. Or, l'innovation technologique n'a pas cessé au courant des dernières décennies. De nos jours, pour tous les systèmes modernes, une connectivité à l'internet est prévue. Les calculs et opérations autour des données sont distribué à différents niveaux de l'architecture du réseau pour optimiser les rendements. On peut voir le contraste par rapport à la Figure 0.2 avec ce réseau moderne d'IIoT illustré dans la figure 0.3 ci-dessous.

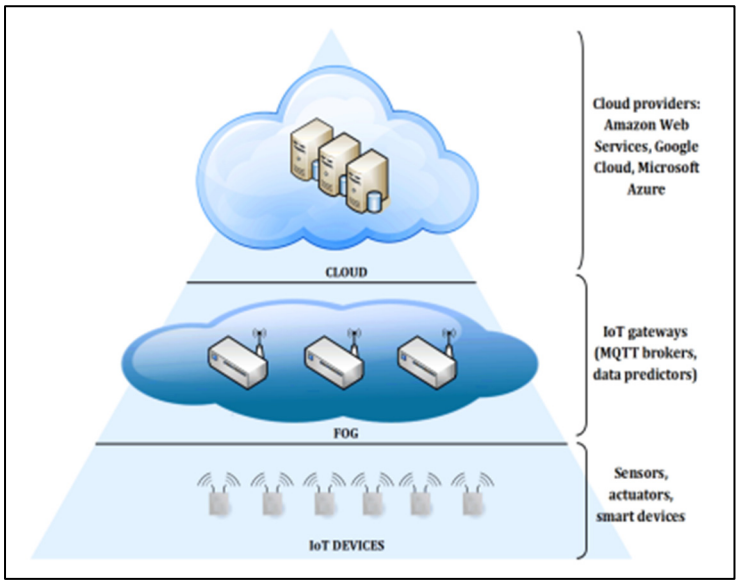

Figure 0.3 Réseau moderne d'internet des objets
Tirée de Peralta et al. (2017)

Comme le montre la Figure 0.3, on retrouve la couche de l'infonuagique au sommet du réseau, contrairement à notre architecture précédente où le plus haut niveau se trouve à être l'ordinateur de configuration ou bien le poste de travail de l'opérateur. On y retrouve aussi un segment dans l'informatique du brouillard, qui est très semblable au concept d'un dispositif de pointe. Sans entrer dans les détails, il faut comprendre que l'architecture a été repensée pour être compatible avec des suites de systèmes infonuagiques très puissants comme Azure et Amazon Web Services. C'est la plus grande différence par rapport aux anciens systèmes, puisqu'il faudra émettre ses données par d'autres protocoles, soit, par exemple, MQTT ou AMQP.

0.2 Énoncé du problème

Dans ce projet de recherche, une solution basée sur l'IIoT est proposée pour permettre la communication entre un robot et son environnement. Cette solution doit apporter une abstraction sur la panoplie de dispositifs et de méthodes de communication qui se trouvent

dans l'environnement opérationnel du robot. Ces dispositifs industriels et domotiques ne sont pas tous interopérables. Les méthodes de communication n'utilisent pas le même protocole de communication. Ainsi, on doit être en mesure de connecter tous ces dispositifs sans craindre un blocage au niveau de la communication entre les dispositifs. De plus, au-delà du flux de données, il faut identifier les fonctionnalités souhaitées dans le contexte opérationnel d'un robot autonome. Ces grandes tâches ont permis l'établissement des requis de ce projet de recherche. Les voici :

- La solution doit pouvoir de communiquer avec plus d'un robot. Il ne faut donc pas se limiter à utiliser exclusivement un contrôleur unique dans l'échange de données;
- La solution doit être en mesure d'utiliser des dispositifs industriels et domotiques usuels incluant des caméras et des capteurs de domotique résidentielle;
- La solution doit permettre au robot d'exécuter ses tâches qui demandent une faible latence de communication.

0.3 Questions de recherche

Des interrogations dégagées de ces requis énumérés seront élucidées par des travaux entrepris dans le cadre de ce projet de recherche. Ces questions sont :

- Question 1: Quel système ou plateforme est en mesure d'intégrer une variété de robots, de dispositifs industriels et domotiques pour la réalisation de ce projet?
- Question 2: Comment gérer une multitude de protocoles de communication et les rendre interopérables?
- Question 3: Comment évaluer la performance d'un tel système pour en déterminer ses limites de fonctionnement?

0.4 Objectifs

L'approche préconisée dans l'obtention de la solution comprend l'étude et l'analyse des technologies réseautiques déployées dans le domaine des passerelles de communication

(*gateways*). En réseautique, les passerelles de communication sont utilisées pour permettre aux paquets de données d'atteindre leur destination en transitant par plus d'un réseau de communication. Il est donc important de comprendre les principes de fonctionnement de ces passerelles pour la conception de notre solution. En ralliant les protocoles industriels et les réseaux IIoT, il est possible de combler l'ensemble des besoins énoncés et répondre aux questions de recherche posées. Les objectifs de ce projet de recherche sont formulés à partir des questions de recherche de la section précédente. Ils sont :

- Objectif 1: Concevoir un système ou plateforme logicielle pouvant communiquer avec une large gamme de robots industriels. L'atteinte de cet objectif permettra de répondre à la première question de recherche;
- Objectif 2: Développer une passerelle de communication afin de permettre à des données de transiter entre différents protocoles de communication. Le développement de celle-ci permettra de bien répondre à la première et à la deuxième question posées;
- Objectif 3: Obtenir les latences et le débit de communication de la solution conçue. Cet objectif permettra de définir les limites opérationnelles de la solution. Les résultats de cet objectif serviront à répondre à la dernière question de recherche de ce projet.

0.5 Contributions

Les principales contributions de ce projet de recherche seront :

- Une plateforme logicielle capable d'intégrer des dispositifs industriels et domotiques à un système robotique par le biais d'une passerelle de communication;
- La détermination des temps de latence de communication par une procédure statistique non paramétrique assure une performance opérationnelle bornée dans un intervalle de confiance.

0.6 Plan du mémoire

Le chapitre 1 de ce mémoire présente la revue de la littérature portant sur les plateformes de développement spécialisées dans le domaine de la robotique, la conception des passerelles de communication et les capabilités des protocoles de communication. Ces plateformes comprennent des fonctions et outils qui sont utiles dans le contrôle et la programmation des robots. Quant aux passerelles de communication, différentes approches de conception ont été étudiées afin de déceler leurs avantages et lacunes. Le chapitre 2 détaille le choix des technologies qui seront déployées dans la conception et le développement de la solution. La conception proprement dite de la solution est présentée dans le chapitre 3. Le chapitre 4 présente le protocole d'expérimentation et l'analyse des temps de latence de la solution.

CHAPITRE 1

REVUE DE LITTÉRATURE

1.1 Introduction

Ce chapitre présente différentes conceptions de passerelles industrielles pour l'Internet industriel des objets (IIoT) recensées de la littérature. Les plateformes de développement et les protocoles industriels de communication nécessaires au développement de ces passerelles industrielles de communication seront également évalués. L'organisation de ce chapitre est comme suit. Premièrement, les passerelles de communication avec un regard tourné vers l'IIoT sont présentées dans la section 1.2. Pour concrétiser la conception de ces passerelles industrielles, il faut des outils de développement appropriés. La section 1.3 présente une analyse des plateformes de développement qui sont disponibles en source ouverte (*open source*). Puisque les passerelles modernes sont multi protocoles, une revue des protocoles IIoT et des réseaux de terrain complètera cette revue de la littérature dans la section 1.4.

1.2 Passerelles industrielles

Comme première passerelle de communication, mentionnons les travaux de Cupek et al. en 2017. L'équipe de Cupek s'est intéressée aux fonctionnalités offertes par le CAN Bus (*Control Area Network*), un protocole particulièrement présent dans les automobiles. Le CAN Bus est un système très utile pour les fabricants de voitures car il est très robuste, capable de prévenir les erreurs de communication et il est peu coûteux (Gurram et al., 2011). Toutefois, ce protocole a des faiblesses. Dans les travaux de Cupek et al., un système LIDAR (*Light Detection and ranging*) pour la détection des distances entre véhicules a été utilisé. Les chercheurs ont identifié une lacune importante de CAN Bus. Si le CAN Bus est très performant dans son utilisation de la mémoire pour la transmission de messages, il ne produit aucune métadonnée sur les messages qu'il transmet (Cupek et al., 2017). Il est donc très difficile d'identifier la provenance des données dans le système. Il est possible d'ajouter des

métadonnées mais cela exige un intermédiaire externe au CAN Bus. Autrement dit les trames de CAN Bus doivent être enveloppées dans un autre protocole de plus haut niveau d'abstraction.

Les travaux de Cupek et al. montrent clairement qu'il serait très difficile d'exploiter des données provenant directement du flux de données du protocole CAN Bus. Pour pallier ce problème, le protocole OPC UA a été utilisé afin d'ajouter du contexte aux données de CAN Bus. Le Protocole OPC UA est standardisé par la norme IEC 62541. (Watson, 2017) Ce protocole joue le rôle de « traducteur » et sert à unifier les systèmes de communication en offrant des architectures de communication variées et est compatible avec plusieurs langages de programmation. (Watson, 2017) L'enveloppement des trames CAN Bus est réalisé par un programme séparé en trois segments pour l'enrichissement des transmissions CAN Bus. Le premier segment accumule les données du système et ajoute des métadonnées en utilisant le schéma d'origine de distribution des données de l'architecture CAN Bus. Le deuxième segment reprend les données du premier segment et les publie dans un serveur OPC UA. L'exploitation des données CAN Bus est alors possible en s'abonnant au serveur OPC. Pour la communication inter CAN Bus, le troisième segment reprend les données du serveur OPC et les publie dans le protocole CAN Bus.

On remarque qu'il est fort possible d'unir différents protocoles de communication afin de pallier les désavantages que certains protocoles présentent. En effet, l'ajout de métadonnées à des messages est une approche intéressante pour la compilation des données pour usages spécifiques en temps différé tels les systèmes d'aide à la décision.

En adoptant le point de vue des objets connectés, l'équipe de Chen a proposé une architecture de passerelle à microcontrôleurs multiples (Chen, 2018). En effet, ils avancent que l'internet des objets peut être appliqué dans une variété de domaines différents, incluant la santé, la logistique, la gestion d'énergie et la supervision de l'environnement. Ces domaines peuvent en effet bénéficier d'appareils connectés afin de créer des services capables de prendre des

décisions à partir des données accumulées. Conséquemment, ils proposent une architecture composée de plusieurs microcontrôleurs afin d'être applicables dans des contextes variés.

On fait mention dans ces travaux de la difficulté d'intégrer différents protocoles de communication ensemble dans un système fonctionnel. Ils soulignent que la conception d'une passerelle de communication efficace passe par une conception judicieuse et l'élimination des goulots de communication. C'est pourquoi ils ont adopté une architecture de type coordonnateur-nœud pour la coordination des microcontrôleurs et le traitement des messages. Dans leur proposition, l'application de microcontrôleurs multiple présente plusieurs avantages:

- Plus de flexibilité et d'évolutivité par rapport à un seul microcontrôleur. La solution est très directe dans ce cas : si on a besoin de plus de puissance, on n'a qu'à ajouter un autre microcontrôleur à l'architecture.
- Un grand niveau d'intégration de cette solution permettra de décentraliser les systèmes qui agissent en temps réel.
- La solution est très peu coûteuse; les microcontrôleurs sont très abordables et rarement en rupture d'inventaire, surtout dans une époque où l'on fait face à une pénurie de puces électroniques.

À plusieurs égards, leur solution proposée peut en effet être très efficace. Elle propose un système flexible et abordable qui peut être utilisé dans une variété de scénarios. Cependant, les applications mentionnées dans l'article de Chen et al. demeurent assez rudimentaires et le Modbus est utilisé pour interrelier les éléments du système multi-microcontrôleur. Il est à se demander s'il est pertinent et souhaitable de coordonner un grand nombre de microcontrôleurs à l'aide d'un bus aussi élémentaire.

Dans le cas de notre passerelle, on planifie l'intégration d'un bras robotique et de caméras dont les signaux seront traités afin d'être exploités par l'algorithme éventuel d'un robot. L'intégration de ces composantes sera très difficile, voire impossible sur un simple microcontrôleur. Si le système multi-microcontrôleur est très économe en terme d'énergie, il

ne fournira pas une puissance suffisante de calcul pour des charges de calcul élevées. De plus, la plupart des équipements industriels utilisent des logiciels qui exigent le soutien d'un système d'exploitation alors que les microcontrôleurs n'en disposent pas. Enfin, il ne semble pas y a voir un moyen facile d'introduire d'autres protocoles de communication dans le système. La passerelle proposée est donc fondamentalement une passerelle mono-protocolaire.

La création d'une passerelle de communication peut être nécessaire dans plusieurs contextes. Par exemple, on voudra effectuer des échanges entre systèmes qui n'ont pas été conçus pour être compatibles. De plus, les médias de transmissions de données peuvent ne pas être suffisants dans certains cas.

Dans l'analyse d'une passerelle de communication effectuée par l'équipe de Khanchuea et Siripokarpirom, ils ont identifié une solution pour étendre la portée des réseaux locaux (Khanchuea, 2019). Celle-ci consiste à déployer une architecture de réseau à multi-saut à l'aide de microcontrôleurs de la famille ESP32 couramment utilisés dans des applications IoT. Un réseau à multi-sauts consiste à utiliser des microcontrôleurs comme intermédiaires pour le trafic des données sans fil (Yi et Shakkottai, 2007). Ainsi, si un des contrôleurs n'atteint pas une composante du système, il peut le faire par l'entremise d'une autre composante.

Le système étudié par Khanchuea et Siripokarpirom comprend aussi un réseau pourvu d'une interface de consultation des données et de la télémétrie par le biais d'un serveur WEB ainsi qu'un broker (courtier) MQTT pour recevoir des messages. Les clients reliés à la passerelle de communication peuvent alors se connecter aux contrôleurs du système.

Cette passerelle a été conçue dans le but d'utiliser différentes technologies de communication telles le réseau Wi-Fi et le Bluetooth à faible consommation d'énergie. Si cette passerelle a répondu à cette question, le potentiel de ce système reste douteux, car l'étude n'a utilisé que six points de communications pour voir l'étendue de son système. Il est donc difficile

d'adéquatement évaluer le potentiel d'une architecture multi-saut en raison de la grandeur de l'échelle rapporté dans cette publication.

Les travaux de Khanchuea et Siripokarpirom ont montré que la communication sans fil peut être un médium efficace pour la transmission de données dans un réseau de communication. Ultimement, le principe d'une connexion réseau peut être utilisé en omettant un point de connexion centrale telle qu'un routeur pour les différentes composantes d'un réseau.

1.3 Plateformes de développement

L'importance d'une passerelle multi-protocolaire a été soulignée dans la section précédente. Le mode d'opération d'une telle passerelle peut être décentralisé pour réduire la congestion du bus de communication. De nos jours, la réalisation d'une telle passerelle industrielle passe nécessairement par des plateformes de développement. Ces dernières offrent des outils et services qui simplifient les tâches de programmation et ainsi accélérent le processus d'implantation. Enfin, les plateformes étudiées sont celles dédiées au développement des systèmes robotisés en concordance avec les objectifs visés de ce mémoire.

1.3.1 LCM

Le système LCM (*Lightweight Communication and Marshalling*) est un protocole de communication sous forme de bibliothèques. Le but principal de LCM est de simplifier les échanges de messages en temps réel. Développé en 2005 au MIT (*Massachusetts Institute of Technology*), ce système offre une infrastructure de messagerie selon le modèle « publication et abonnement » (*Publish/subscribe*). Plusieurs systèmes robotiques opérant au sol, dans les airs et sous l'eau utilisent le LCM comme moyen de communication. Le système semble suffisamment performant pour soutenir une telle diversité d'applications robotiques. Le noyau de LCM repose sur trois concepts fondamentaux (Huang et al., 2010) et est le sujet des sous-sections suivantes.

1.3.1.1 Spécification des types

Les données dans ce système de messagerie sont toujours typées. Pour réaliser correctement les échanges de données, le LCM utilise une méthode de définition des types qui est basée sur les types du langage de programmation C. Concrètement, les types de données composées sont encapsulés dans des structures du langage C. Ainsi, le nombre et l'alignement des octets d'un type de données sont spécifiés directement dans un fichier d'entête du code source. Cette spécification des types est partagée entre les abonnés et l'émetteur des messages avant le début de la communication.

1.3.1.2 Sérialisation (Marshalling)

Pour pouvoir transmettre les données via un réseau de communication (filaire ou sans-fil), le système LCM doit convertir les données en format série. L'encodage résultant est une suite de symboles qui pourraient être transportés par le réseau. Les symboles utilisés dans la sérialisation de LCM sont binaires suivant la convention du langage C et de la norme IEEE 754 pour des nombres en virgules flottantes. Pour contrôler les erreurs de réception, chaque transmission est précédée d'une « empreinte » qui est le résultat d'un hachage incluant le type et le nom des données envoyées.

1.3.1.3 Communication

La transmission des messages est basée sur le protocole UDP (*User Datagram Protocol)* multidiffusion (*multicast*). C'est le protocole UDP qui permet au système LCM de réaliser le modèle communication publication/abonnement. En effet, la nature unidirectionnelle de la multidiffusion rend inopérant le protocole TCP (*Transmission Control Protocol*) et, par le fait même, les mécanismes de retransmission ne sont pas disponibles. Un participant, soit un sous-système, un équipement ou un composant, peut recevoir tous les messages publiés. C'est au participant de filtrer les messages reçus et de retenir ceux qui lui sont destinés. Enfin, le

système LCM dispose d'outils pour observer et analyser la transmission et la réception des messages.

1.3.1.4 Avantages et désavantages

Le système LCM est composé d'un ensemble de bibliothèques de programmation. Il est donc facile de l'incorporer dans un projet de développement. D'ailleurs, il existe un bon nombre de tutoriels et réalisations pour aider à la compréhension et au démarrage de projet. De plus, l'utilisation du protocole UDP multidiffusion réduit la latence dans la transmission des messages. En revanche, la définition des types basée sur les types du langage C rend difficile la compatibilité du système vers d'autres langages de programmation. De nos jours, l'utilisation des notations formelles telles le XML (*Extensible Markup Language*), XHTML (*Extensible HyperText Markup Language*) et même le JSON (*Javascript Object Notation*) peut pallier le problème de la portabilité de la définition des types. Or, cela exigerait une refonte majeure du système LCM. Dans son état actuel, le LCM accepte des données composées de types « tableau » et « structure ». Ici, un tableau est une suite contiguë d'un même type de données et une structure est un regroupement de différents types de données. Or, dans bien des applications modernes, on peut recourir à des données composées beaucoup plus versatiles. Par exemple, le dictionnaire est un type associant une clé et une valeur. La clé peut être une chaîne de caractères tandis que la valeur peut être n'importe quel type de données incluant un autre dictionnaire. Malheureusement, ces types de données modernes ne sont pas disponibles dans le système LCM.

1.3.2 ROS

En 2007, la communauté des logiciels ouverts (*Open Source*) a commencé à développer un système qui est à la fois cadriciel (*framework*) et intergiciel (*middleware)*. Ce système est baptisé ROS (*Robotic Operating System*). Le principe de fondation de ROS durant son développement est de « cesser de réinventer la roue » afin de permettre à différents composants

de facilement se rattacher à celui-ci pour le développement logiciel de systèmes robotiques. Parmi ces composantes, on y retrouve notamment une variété de bras robotiques, des caméras 2D et 3D ainsi qu'une panoplie de capteurs. Au travers des années, des fabricants de robots tels qu'ABB, FANUC et Universal Robots ont créé des « paquets » ROS (*packages*). Ces packages permettent aux composantes d'être interfacés et d'être intégrés à l'ensemble de ROS.

Du point de vue opérationnel, le système ROS est compatible avec une variété de langage de programmation, tels le C++, Python, Octave et Lisp. Tout comme le système LCM, la communication au sein de ROS est réalisée par le modèle publication/abonnement. De plus, le système ROS intègre des outils pour le surveillance des transmissions, pour la visualisation de données et pour la simulation des robots (Quigley et al., 2009). Pour pouvoir déployer l'environnement de développement ROS, il est nécessaire de le construire à partir de son code source. Cette construction est en fait une compilation des fichiers sources permettant au ROS de s'adapter au modèle du robot, aux périphériques physiques du système et aux composants logiciels désirés. Les commandes « catkin » sont l'outil principal de ROS pour construire différents environnements de développement (Allen, 2016). Le système ROS possède un noyau principal qui est essentiel au fonctionnement d'une application (Kouba, 2016). La création du noyau est nécessaire puisque tous les sous-systèmes de ROS se rattachent à lui. Le noyau établit également un serveur de paramètres qui permet à l'utilisateur de retrouver différents paramètres du système. Son rôle est similaire à des variables globales dans l'ensemble du système ROS. Ainsi, l'utilisateur peut lancer des commandes pour obtenir le contenu des paramètres du système.

D'un point de vue de communication, le système ROS est organisé en maître et nœuds. Le maître a pour fonction de rattacher tous les nœuds, topics et autres éléments afin qu'ils puissent communiquer les uns aux autres. Si l'utilisateur le désire, le maître établit également un « rosout », une sorte de journal de l'ordinateur, qui décrit les activités qui se passent au travers du maître. La Figure 1.1 montre quelques éléments de communication rattachant à un maître ROS. Ces éléments sont présentés dans les sous-sections suivantes.

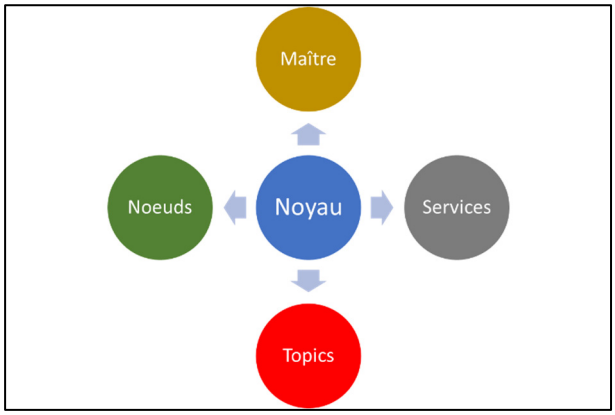

Figure 1.1: Illustration de quelques éléments rattachés au noyau ROS

1.3.2.1 Nœuds ROS

Un nœud possède généralement un rôle spécifique dans une application, qu'il soit physique ou virtuel. Par exemple, un nœud peut prendre en charge un capteur qui doit lire une quantité physique et la traiter. De même, un nœud peut également jouer le rôle d'un serveur qui a pour fonction de transmettre des données à un autre nœud. On peut également lui assigner des rôles spécifiques. En structurant convenablement l'application, on peut répartir des tâches distinctes aux nœuds. L'ensemble des nœuds associés à un maître forme une arborescence modulaire permettant aux développeurs d'y apporter leurs contributions ciblées à des nœuds spécifiques. Enfin, ce principe de développement modulaire est ancré dans la philosophie des logiciels ouverts.

1.3.2.2 Topics ROS

Les topics ROS sont un mécanisme de base de la communication inter-nœuds. C'est un type de communication unidirectionnelle où un nœud publieur transmet le sujet comprenant un titre dans un format de message spécifique (Quigley et al., 2015). Les nœuds doivent s'abonner aux topics pour recevoir les messages envoyés par le publieur. La nature des messages est

déterminée par le sujet du topic, mais la fréquence de transmission ainsi son contenu sont déterminés par le publieur. On peut conceptualiser un topic comme un canal de communication dédié à un sujet. Notons qu'un publieur peut créer plusieurs topics et que les abonnés peuvent aussi s'abonner à plus d'un topic. Clairement le mécanisme des topics sert à créer une relation multivoque entre les publieurs et les abonnés.

1.3.2.3 Services ROS

Sous le système ROS, l'appel d'un service est similaire à un appel de fonction d'une bibliothèque. Cependant, le fonctionnement d'un service est différent de celui d'une fonction de programmation. Un service est une fonctionnalité dans un nœud. Son appel fait exécuter une ou des tâches dans un nœud et il est possible d'y recevoir des réponses. L'appel de service est donc plus étendu qu'un échange de messages entre les nœuds puisqu'il donne lieu à des traitements de données. Pour permet l'appel de service, le système ROS utilise un langage de description de service (srv) pour définir comment un service est appelé (*service request*) et son type de retour (*service response*). Ainsi, on doit préparer soigneusement les descriptions de service lors de la compilation de l'environnement de développement ROS par les commandes « catkin ». Une fois la description du service est en place, il ne manque que deux étapes pour que le service soit utilisable. La première est la déclaration et le nom d'un serveur qui sera créé par le nœud à partir du srv. Ce faisant, le serveur saura comment le service est appelé et le type de retour de sa réponse. Le nom du serveur est alors enregistré dans le noyau ROS. La deuxième étape est de déclarer un client pour le service. Il est généralement recommandé de vérifier l'existence du service avant de faire une requête à ce dernier. À noter que ces clients, tout comme les abonnés d'un topic ROS, peuvent utiliser plus d'un service. En revanche, il ne peut y avoir qu'un seul serveur portant le même nom.

L'appel de ces services est fait de manière bloquante. Lors d'un appel de service, le nœud appelant est bloqué jusqu'à la réception de la réponse du service. Il devient alors facile de contrôler une séquence d'actions synchronisées. En revanche, un temps de service trop long

peut empêcher le bon fonctionnement d'un nœud. On peut aussi envisager la possibilité d'interblocage (*deadlock*) où des nœuds sont bloqués en s'attendant mutuellement. Il faut alors éviter de programmer des algorithmes de longue durée dans les nœuds et introduire, au besoin, un temps d'expiration (*timeout*) dans les algorithmes pour éviter l'interblocage.

1.3.2.4 Packages ROS

Sous ROS, les éléments des sections 1.2.2.1 à 1.2.2.3 sont rattachés à une entité appelée « paquet » (*package*). Un package ROS est généralement associé à un composant physique tel un capteur ou un actionneur. Dans un package, on retrouve des nœuds, des topics et des services conçus dans un but spécifique. C'est donc en quelque sorte un assemblage d'éléments du système ROS possédant des fonctionnalités utiles et réutilisables. Deux (2) fichiers sont utilisés pour décrire le contenu d'un package : i) « CmakeLists », qui est nécessaire à la compilation des codes sources du package; ii) « Package.XML », qui est un markup contenant le nom, la version, le nom des auteurs et autres propriétés du package ROS qui sont utiles pour les commandes catkin. Les fabricants de produits robotiques, les développeurs et les chercheurs peuvent contribuer au développement en produisant des packages ROS. C'est pourquoi la popularité de ROS a augmenté de manière exponentielle, tout comme le nombre de package disponibles. (Boren et al., 2011)

1.3.2.5 Avantages et désavantages

En soi, ROS est un cadriciel et intergiciel très intéressant pour la programmation et le contrôle des robots. Il favorise la réutilisation du code dans les applications et offre un nombre impressionnant de packages sous licence open source. De plus, ROS a plusieurs outils de simulation et de visualisation pour le débogage. Cependant, il est très difficile de réaliser des traitements en temps réel sous ROS. Ce dernier est essentiellement un système Linux de distribution Ubuntu et l'accent a été mis sur l'exécution multitâche des programmes. Aussi,

l'installation et la création d'un environnement de développement nécessitent de bonnes connaissances en informatique, ce qui n'est pas toujours le cas des roboticiens spécialisés.

1.3.3 MRPT

MRPT (*Mobile Robot Programming Toolkit*) est un cadriciel ayant une mission similaire aux systèmes ROS et LCM. Pour accomplir sa mission, le système MRPT a adopté l'approche dite « d'intégration ». En effet, les fonctions utiles dans des applications robotiques sont directement intégrées dans le système sous forme de « modules ». Ainsi, des fonctions en statistique, en traitement du signal, les quaternions et le déterminant du jacobien sont déjà inclus dans le système. Des fonctions de haut niveau telles la cartographie des lieux et la localisation simultanée, des filtres de Kalmen, l'inférence Bayésienne et la planification de trajectoires sont également disponibles (Tuza et al., 2010).

Le système MRPT est distribué sous forme de bibliothèques de programmation C++. Des interfaces de programmation d'application (*API*) ont été créées pour être compatibles avec le langage de programmation Python. De plus, le système MRPT est maintenant disponible sous forme d'un package ROS. Il est donc possible d'utiliser les capacités de MRPT à l'intérieur du système ROS.

1.3.3.1 Avantages et désavantages

Avec une panoplie de fonctions de base et de haut niveau, le système MRPT peut accélérer grandement le développement des applications robotiques. De nature multiplateforme, le système peut être utilisé sous Windows et Linux. Les capacités de ce système sont remarquables et elles sont orientées vers la robotique mobile autonome. Une grande importance est accordée aux procédures et techniques réalisant la cartographie des lieux et la localisation simultanée. L'ajout de nouvelles capacités dans le système MRPT passe par la compréhension générale de l'interface des modules. Cette interface n'a pas été rendue

publique. Ainsi, on ne peut pas l'étendre ou l'adapter pour rendre en un système dédié MSA (*Mission-Specific System*).

1.3.4 Node-RED

Node-RED est un outil de développement open source multiplateforme d'IBM basé sur le flux de données (Lekić et al., 2018). Cet outil visuel offre un grand nombre de modules appelés nœuds et son principe de fonctionnement consiste à relier les nœuds pour permettre le flux des données d'une source vers sa destination à l'aide de liens visuels. La Figure 1.2 est un exemple visuel de cet arrangement. Dans le schéma de la Figure 1.2, les boîtes sont des nœuds et les liens représentent des connexions entre les nœuds. Tout comme un système physique, les données de Node-RED passent de la sortie des nœuds vers l'entrée des nœuds. Le flux de données débute dans la boîte nommée « temperature » et passe simultanément dans les nœuds « Display temperature in dashboard », « Prepare data for storage » et « If temperature is abnormal ». Dans ce dernier nœud, si la température est en-dehors d'une plage de valeurs, réglées à priori dans la configuration du nœud, alors les données enclencheront les nœuds « Send SMS » et « Send email ». Dans Node-RED, on peut donc réaliser des transformations séquentielles, parallèles et effectuer des prises de décision.

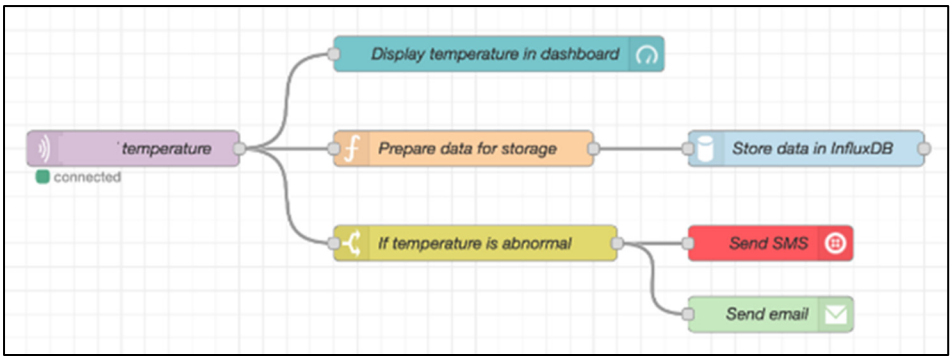

Figure 1.2 Exemple d'un flux dans Node-RED

1.3.4.1 Architecture

Node-RED est basé sur l'environnement d'exécution « Node.js ». C'est un environnement JavaScript qui permet l'exécution des micro-services par des fils d'exécution (*threads*). Il utilise le port 1880 de l'ordinateur local pour communiquer avec le fureteur WEB et afficher son interface utilisateur. Reposant sur le paradigme des flux de données, l'architecture de Node-RED est pilotée par des données (*data-driven*). C'est-à-dire que le système réagit aux entrées de données et à leurs déplacements parmi les modules. Par exemple, lorsqu'une donnée est entrée dans l'interface d'utilisateur, Node-RED reçoit cette donnée en arrière-plan et la transmet au nœud destinataire approprié. Le nœud récepteur effectue la tâche programmée, transmet le résultat au nœud suivant et ainsi de suite.

1.3.4.2 Nœuds Node-RED

Node-RED comprend des nœuds de programmation. Chaque nœud est responsable d'exécuter une tâche spécifique et chaque lien indique une connexion de données entre deux nœuds. Chaque nœud possède aussi une configuration dans laquelle des paramètres modifient le comportement du nœud. Node-RED dispose d'un bon de nombre de nœuds dans sa bibliothèque de base. On trouve dans cette bibliothèque des nœuds pour déclencher les flux de données, des nœuds de calcul et des nœuds de sorties pour afficher des résultats. On peut également joindre du code Javascript dans le flux par le biais d'un nœud de fonction, ce qui procure à Node-RED une grande flexibilité dans le traitement de données. D'ailleurs, un nombre de nœuds, conçus par des fabricants de produits industriels tels Allen Bradley, Siemens, Mitsubushi, Omron et Schneider, a été mis à la disposition des utilisateurs. Conséquemment, plusieurs protocoles industriels sont disponibles sous forme de nœuds dans Node-RED.

1.3.4.3 Flux

Les flux Node-RED ont été illustrés jusqu'à présent comme une chaîne de nœuds accomplissant des tâches désirées. Or, il est possible d'imbriquer les flux Node-RED dans un nœud et les faire exécuter d'une façon concurrente dans des fils d'exécution. Dans l'éditeur de Node-RED, on peut visualiser les flux imbriqués en activant le nœud similaire à l'ouverture d'un nouvel onglet dans un navigateur WEB. Cependant, plus le nombre de fils d'exécution est grand, plus le niveau de complexité du flux augmente et la gestion de ces fils d'exécution peut devenir très difficile (Tilkov, 2010). Enfin, les flux peuvent être enregistrés et partagés dans un fichier de format JSON (*JavaScript Object Notation*).

1.3.4.4 Avantages et désavantages

Node-RED est un outil très utile pour concevoir des programmes sans être un expert en programmation. L'Éditeur graphique est très utile pour visualiser l'état des flux et pour le débogage. Cependant, il n'est pas possible de modifier un nœud produit par de tierces parties. En effet, si les paramètres disponibles dans la configuration d'un nœud ne suffisent pas à obtenir le résultat désiré alors ceci pourra causer des problèmes. Dans le cas des protocoles industriels, il faut parfois recourir à l'essai-erreur pour diagnostiquer le problème. Un autre problème rencontré est au niveau de la qualité de fonctionnement et de la fiabilité des nœuds disponibles en open source. Il faut soumettre ces nœuds à des tests de fonctionnement et de performance pour s'assurer qu'ils rencontrent réellement les spécifications proclamées.

1.4 Protocoles de communication

Une passerelle de communication sert d'intermédiaire entre les équipements industriels et l'Internet. Il est donc important d'identifier ces protocoles de communication pour une analyse éventuelle de leurs caractéristiques. Dans cette partie de la revue de la littérature, les protocoles couramment utilisés dans le domaine de l'IIoT et les protocoles constituant les réseaux de terrain (*Fieldbus*) sont abordés.

1.4.1 Zigbee

Le protocole Zigbee est souvent utilisé dans des projets impliquant l'internet des objets (IoT). Il est spécialisé dans l'envoi de données à basse fréquence et ne consomme que très peu d'énergie. En effet, l'émission de données à basse fréquence peut être avantageux lorsque les données collectées par des capteurs ne sont requises que lorsque demandées. Ceci fait en sorte que des capteurs miniatures peuvent optimiser leur consommation d'énergie.

Le Zigbee peut accueillir une grande quantité de capteurs indépendants dans un seul réseau de communication. Étant donné que les capteurs utilisent une fréquence radio modulable et qu'un groupe de fabricants se sont regroupés pour produire des composants compatibles, de nouveaux standards ont été établis en fonction de ce protocole (Ramya et al., 2011). Ces capteurs, qui servent d'appareil dans un réseau Zigbee, sont reliés à des routeurs, qui sont à leur tour orchestrés par un coordonnateur qui contrôle les fréquences d'envois ainsi que les fréquences désirées par le réseau de l'infrastructure. Enfin, différentes topologies sont possibles et peuvent être préférables selon les scénarios, étant donné que Zigbee est un protocole propriétaire sans fils contrairement à la plupart des protocoles qui se fient à une connexion TCP/IP pour échanger de l'information.

1.4.1.1 Avantages et désavantages

Zigbee est un protocole très efficace en termes de consommation d'énergie (Firdaus et al. 2014). En effet, sa faible consommation permet d'augmenter la durée d'utilisation des systèmes fonctionnant sur batteries. De plus, Zigbee possède une variété de pilotes logiciels qui lui permettent d'être utilisé dans un scénario de passerelle de communication. Malgré le fait qu'il utilise son propre système de radio fréquence afin de transmettre des données. Cependant, cette faible consommation d'énergie vient aussi avec une moins grande capacité de performance, comparativement au Wi-Fi, par exemple. En effet, sa portée et son débit de

transmission sont inférieurs comparé au Wi-Fi, qui utilise la transmission de données par TCP/IP.

1.4.2 LoRa

LoRa est une autre technologie LPWAN (*Low Power Wide Area Wireless Network*) est un autre protocole basé sur des ondes de radio. C'est un compétiteur direct à Zigbee en matière de connectivité pour établir l'immense interconnectivité qui sera nécessaire dans un futur proche (Noreen et al. 2017). En effet, la même étude estime que LoRa occupe environ un quart du marché des 30 milliards d'appareils connecté dans l'univers de l'internet des objets. En termes de technologie, ses capacités sont très similaires à celle de Zigbee, ayant la même mission de pouvoir soutenir des capteurs qui feront le traitement des données localement avant de les émettre vers une passerelle unique compatible avec les ondes radio que les capteurs émettent. Ces ondes radio provenant des capteurs ont été conçues de façon à ignore le bruit de l'environnement et à transmettre la valeur exacte qu'elle veut transmettre à sa passerelle.
Comme mentionné initialement, les capteurs qui servent de sources de données aux systèmes émettent des informations prétraitées et les soumettront à une passerelle pouvant convertir le signal. Ces passerelles de LoRa convertiront ensuite la charge utile (*payload*) des messages émis en un format envoyable par TCP/IP vers le serveur qui peut orchestrer de multiples passerelles. Ces serveurs occupent un rôle similaire à celui des coordonnateurs de Zigbee, chacun ayant leur terminologie spécifique. La clé pour augmenter la portée de LoRa réside dans la gestion de la répartition des passerelles selon les points de données qu'elle doit aller chercher.

1.4.2.1 Avantages et désavantages

En tant que protocole de faible puissance, LoRa peut être utilisé dans des applications à faible consommation d'énergie. Son plus grand atout est toutefois sa portée. Un réseau de LoRa peut facilement s'étendre sur des centaines de mètres (Bor et al., 2016). Ceci est donc très

intéressant dans les réseaux ruraux et les zones restreintes d'accès, car les interventions matérielles sont beaucoup moins fréquentes comparativement à une usine par exemple. LoRa possède également une bonne fiabilité en ce qui concerne la réussite du taux d'envoi de messages. Toutefois, LoRa ne possède pas un bon débit pour pouvoir transmettre une variété de messages. La latence d'envoi de message est également très lente, pouvant même atteindre les 5 secondes (Bor et al., 2016).

1.4.3 MQTT

MQTT (*Message Queueing Telemetry Transport*) est un protocole de communication qui a vu le jour en 1999. Depuis les dernières décennies, le protocole a pris beaucoup d'ampleur dans la communication de type « machine à machine » et dans des applications IoT (Mishra et al., 2020). OASIS Open, une société de programmation open source a mis en vedette ce protocole en 2013 et l'a standardisé dans le domaine de l'IoT.

Le fonctionnement de MQTT est basé sur le modèle de publication et abonnement par l'intermédiaire d'agents appelés *Brokers* qui jouent le rôle de serveurs pour transmettre et recevoir les messages. Le broker utilise le transport TCP/IP et conséquemment, le MQTT peut fonctionner sur des réseaux filaires et des réseaux sans fils tels le Wi-Fi et LTE (*Long Term Evolution*). Afin d'optimiser la transmission de messages, des niveaux de qualité de services sont disponibles aux applications utilisant le protocole MQTT.

Les relations de base d'une communication par MQTT sont illustrées dans la Figure 1.3. On retrouve les quatre éléments principaux de MQTT soit le serveur, le client, le publieur et l'abonné.

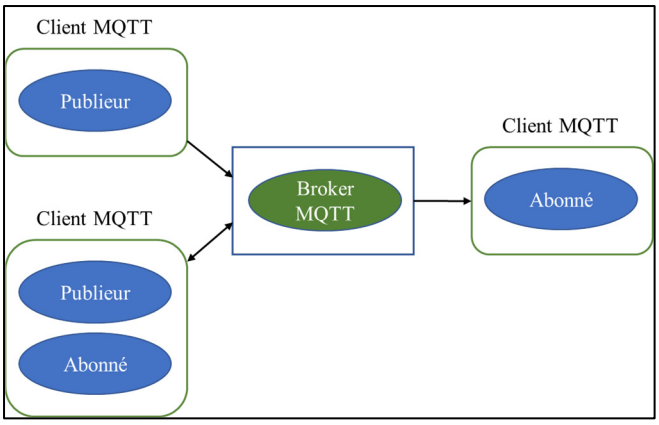

Figure 1.3 Relations logiques des
principaux éléments de MQTT

À l'instar du système ROS, les flèches représentent la direction de communication des topics. L'élément important dans le fonctionnement de MQTT est le broker qui réside au sein de cette architecture. Le broker sert en fait de bureau de poste pour tous les messages de MQTT et permettra de distribuer et de recevoir ces messages. De cette façon, les abonnés n'ont pas à connaître l'adresse IP des publieurs. Ils doivent seulement connaître l'adresse IP du broker. Ce fait est important, car cela permettra aux publieurs et aux clients d'être protégés par des pare-feux sans visibilité à l'extérieur de leurs réseaux locaux. Pour qu'un publieur puisse envoyer des messages, il faut qu'il identifie un topic MQTT qui servira de canal de transmission. Contrairement aux topics ROS, la nature de l'information n'a pas besoin d'être définie et peut-être dans un format quelconque.

Associé à la transmission des messages MQTT est le niveau de qualité du service. Ce dernier peut être considéré comme le niveau de certitude requise pour la transmission des messages par le broker. Par exemple, le niveau 0 de qualité signifie que le message ne sera envoyé qu'une seule fois et qu'aucune action ne sera faite au-delà de l'envoi du message. Il n'y a donc pas de procédure qui vérifie si on a envoyé le message. Le niveau 1 ajoute l'étape de vérification de la réception du message par le broker. Le niveau 2 fait la même chose et va même jusqu'à la vérification de l'acheminement du message jusqu'à l'abonné. L'important à retenir est que la

bonne pratique dans le cas d'une publication et du niveau de la qualité est purement contextuelle. Il faut donc faire le bon choix en fonction du contexte lorsque l'on effectue la génération d'un publieur sur un réseau MQTT.

1.4.3.1 Avantages et désavantages

Tout comme le système de communication de ROS, le protocole MQTT réalise la relation multivoque entre les publieurs et abonnés. De plus, le MQTT a été sélectionné par Microsoft pour son infrastructure infonuagique Azure (Gemirter et al., 2021). La portée de cette décision rend ce protocole encore plus visible auprès des entreprises et des développeurs. Cependant, l'introduction de brokers comme agents intermédiaires augmente considérablement la latence dans la transmission des données. De plus, le niveau de chiffrement des données dans MQTT est considéré comme inférieur par rapport à d'autres protocoles (Mishra et al., 2020).

1.4.4 Protocoles industriels

Dans l'industrie manufacturière, les réseaux de terrain (*Fieldbus*) sont un regroupement de protocoles de communication dédiés aux équipements et dispositifs industriels. La majorité de ces protocoles sont réalisés par des technologies filaires. Au Québec, les réseaux de terrain sont omniprésents dans la production des biens et services en raison de l'automatisation obligatoire de ces secteurs d'activité économique. À travers leur longue histoire, les réseaux de terrain ont eu de nombreux ajouts et modifications technologiques. Le résultat est la fragmentation du marché des réseaux de terrain dominés par quelques fabricants et équipementiers. Par exemple, le FINS (*Factory Interface Network Service*) de la compagnie Omron et Ethernet/IP (*Industrial Protocol*) de l'entreprise Rockwell sont des protocoles *de facto* intégrés dans les produits vendus par ces entreprises. Ces réseaux de terrain ne sont pas compatibles sans un agent intermédiaire qui peut réaliser la traduction des protocoles.

Le FINS est un protocole propriétaire créé par la compagnie Omron. Il utilise le protocole TCP/IP pour transmettre les données sur le réseau. L'échange et la structure des données sont basés sur un adressage associé à des adresses mémoires. Pour pouvoir recevoir des données, il faut spécifier les bonnes adresses et le bon format de données (Hashim et al., 2007).
Ethernet/IP (*Industrial Protocol*) est un aussi protocole propriétaire qui a été créé par la compagnie Rockwell Automation. Contrairement à FINS, les données sont elles-mêmes identifiables sans avoir recours à des adresses mémoires. Ceci rend le protocole adéquat pour des applications de contrôle, de configuration et d'acquisition de données (Brooks, 2001).

On constate que chaque protocole industriel possède des caractéristiques distinctes et la compatibilité de ces protocoles n'est pas la préoccupation première des équipementiers. Pour l'industrie manufacturière, cela peut causer des problèmes techniques majeurs. L'interopérabilité des équipements en provenance de différents équipementiers n'est pas possible sans l'ajout d'agents de traduction. Ces derniers représentent des coûts d'investissement supplémentaires. Reconnaissant cette situation difficile, un consortium d'industriels a formé la fondation OPC (Open Platform Communications) afin de promouvoir la compatibilité des réseaux de terrain et la standardisation des agents de traduction.

OPC UA (*Open Platform Communication Unified Architecture*) est le résultat des travaux de cette fondation. Il s'agit d'une architecture de communication standardisée pour plus de 60 types d'équipements industriels. En autres, cette architecture comprend les spécifications des :

- Méthodes d'authentification et de chiffrement;
- Modèles de communication client-serveur et publication/abonnement;
- Conversion des protocoles de terrain vers les protocoles Internet.

À l'heure actuelle, des équipements pilotés par l'OPC UA sont déployés dans l'industrie mais ils n'ont malheureusement pas encore reçu l'approbation générale. Les systèmes munis de réseaux de terrain traditionnels sont parmi les plus nombreux dans l'industrie manufacturière (Eckhart et al., 2018).

1.5 Conclusions

On a fait une revue de différentes passerelles industrielles, de plateformes de développement ainsi que des protocoles de communication. Pour chacune des sous-sections, des avantages et désavantages ont été constatés selon le concept posé. Ces différents concepts et technologies ne représentent pas l'entièreté des choix possibles à l'égard de la conception d'une passerelle de communication. Cependant, il est important de retenir que ces choix initiaux sont importants, car ceux-ci influenceront les développements qui découleront une fois que l'architecture est établie.

CHAPITRE 2

CHOIX DES TECHNOLOGIES

2.1 Introduction

Dans ce chapitre, on établira les choix quant aux technologies à déployer dans le projet parmi celles présentées dans le chapitre 1. On fera le point sur les technologies retenues pour faire partie de la passerelle. L'ensemble de ces choix constitueront alors un modèle de communication qui sera concrétisé dans le cadre de cette recherche. Dans ce chapitre, la section 2.2 illustrera les choix quant aux plateformes de développement sélectionnées pour faire partie du système. Tout comme les plateformes de développement, la section 2.3 analysera le choix des protocoles de communication qui feront partie du système étudié.

2.2 Choix de la plateforme de développement

Une plateforme de développement logiciel est un environnement permettant l'utilisation des services applicatifs et dans une certaine mesure facilite la gestion de projet (Wang et al., 2015). Quatre plateformes ont été présentées dans le chapitre 1 « Revue de la littérature » et elles sont :

- LCM (*Lightweight Communication and Marshalling*);
- ROS (*Robot Operating System*);
- MRPT (*Mobile Robot Programming Toolkit*);
- Node-RED.

Chacune de ces plateformes possède des caractéristiques très intéressantes selon le point de vue de langages de programmation, de la capacité de communication et de la disponibilité des services applicatifs en robotique. Évidemment, les objectifs de recherche énoncés dans la section 0.4 de ce mémoire serviront de guide dans la sélection de la plateforme de développement. Ainsi, la plateforme de développement doit être compatible avec les

contrôleurs de différents modèles de robot. Elle doit pouvoir communiquer avec des équipements industriels tels des automates programmables, des caméras, des capteurs et des actionneurs. Cela sous-entend qu'elle possède aussi la capacité de transmettre et de recevoir des données venant des réseaux de terrain. La compatibilité avec différents robots est un critère important afin de réduire le temps de développement nécessaire. En effet, la conception et le développement des contrôleurs de robot constituent un sujet de recherche en soi. Il en est de même pour les équipements industriels. Développer des pilotes qui s'interfaceront aux capteurs et actionneurs est une tâche fastidieuse qui n'apporte pas de valeur ajoutée à ce projet de recherche. Le Tableau 2.1 présente le résultat de l'analyse des critères de sélection.

Tableau 2.1 Critères de sélection appliqués aux plateformes de développement

	LCM	ROS	MRPT	Node-RED
Compatible aux contrôleurs de robot	X	√	√	√
S'interface à des équipements industriels	X	√	X	√
Communique avec des réseaux de terrain	X	√	X	√

L'application des critères de sélection révèle que le système LCM n'offre aucun service applicatif pour les trois critères de sélection. Le système MRPT rencontre le critère de la compatibilité aux contrôleurs de robot[1] mais pas l'interfaçage à des équipements industriels ni la communication avec les réseaux de terrain. Les plateformes ROS et Node-RED ont, quant à elles, rencontré tous les critères de sélection.

Dans le contexte de ce projet de recherche, le système ROS semble être la meilleure plateforme pour concevoir et développer des applications impliquant des bras robotisés. Sa compatibilité avec un grand nombre de bras manipulateurs est un facteur déterminant dans cette décision.

[1] Cette compatibilité est assurée par une autre plateforme appelée « Open Mobile Robot Architecture », laquelle agit comme la couche fondation de MRPT.

Le système Node-RED présente lui aussi une compatibilité avec des contrôleurs, mais cette capacité est davantage orientée vers des robots amateurs du genre DIY (*Do-It-Yourself*). La sélection du système ROS procure aussi un avantage en matière de conception d'applications robotiques : ses APIs sont multilingues et possèdent un modèle de communication très développé.

Le constat est quelque peu différent pour les réseaux de terrain. La disponibilité des réseaux de terrain est nettement supérieure pour le système Node-RED en comparaison au système ROS. L'industrie d'automatisation et les équipementiers ont contribué énormément au développement des nœuds Node-RED compatibles avec leurs équipements et les réseaux de terrain associés. L'utilisation de Node-RED dans le développement de la passerelle de communication allégera grandement le travail d'interfaçage des équipements et de la communication des données à l'aide des réseaux de terrain. De plus, Node-RED utilise un modèle de programmation graphique pour le prototypage rapide et le cycle de développement d'une application s'en trouve grandement accéléré. C'est pourquoi le Node-RED est aussi retenu comme plateforme de développement pour la passerelle de communication.

2.3 Choix des protocoles de communication

L'un des dénominateurs communs des systèmes ROS et Node-RED est la capacité de communication en réseau. Cette capacité permet à ces systèmes de coordonner les activités au sein même de la plateforme. Elle permet aussi à des applications développées de contrôler divers équipements et dispositifs à distance. Il y a donc deux catégories de communication en réseaux qui sont en jeu. La première catégorie est représentée par des réseaux locaux et Internet. Dans cette catégorie, les données sont transmises aux destinataires et le délai de réception peut varier d'une transmission à l'autre en fonction du niveau de congestion dans le canal de communication utilisé. Les réseaux locaux dédiés peuvent réduire considérablement le délai de réception des données. Cependant, en franchissant la connexion MODEM de l'entreprise, les paquets de données deviennent la responsabilité des fournisseurs de service WEB. On ne peut plus exercer un contrôle sur le délai de réception des données. Autrement

dit, pour cette catégorie de réseaux, le délai de communication ne doit pas être le seul critère de sélection.

2.3.1 Comparaison entre MQTT, Zigbee et LoRa

Le protocole MQTT est implanté dans des applications IIoT (Katsikeas et al., 2017). Plusieurs fabricants industriels offrent des clients MQTT au même titre que le port TCP/IP dans leurs produits. Le Zigbee, quant à lui, est un protocole développé pour le sans-fil à courte portée. Il peut être converti en réseau TCP/IP au moyen d'une passerelle dédiée. Ce besoin supplémentaire de Zigbee rend le système de communication plus complexe et plus coûteux à réaliser. On peut donc rejeter d'emblée l'utilisation de Zigbee comme réseau de communication pour le contrôle des équipements et dispositifs à distance.

Quant au protocole LoRa, ce protocole sans fil est spécialisé dans la couverture de grande distance. L'avantage principal de LoRa réside dans sa longue portée ainsi que sa basse consommation d'énergie. En revanche, la longue portée de LoRa est obtenue à l'extérieur et en plein air. De plus, l'émetteur et le récepteur sont normalement alignés et placés en hauteur. Dans le contexte d'une application industrielle, les équipements et dispositifs sont souvent placés à l'intérieur dans un environnement bruité électromagnétiquement. Il est difficilement concevable que le rayonnement de LoRa puisse pénétrer dans un endroit aussi achalandé qu'une usine de production comportant de nombreuses sources d'interférence.

Pour ces raisons, le protocole MQTT a été sélectionné comme protocole de communication dans la conception de la passerelle. Mentionnons que les plateformes de développement ROS et Node-RED disposent de clients MQTT prêts à être utilisés.

2.3.2 Réseaux de terrain

La deuxième catégorie de communication est représentée par des réseaux de terrain. Ces réseaux ont une latence de communication très faible afin d'accommoder la nature « temps

réel » des échanges de données. Cela signifie qu'il existe une forme de gestion dans l'utilisation du canal de communication. La gestion couramment rencontrée est la technique dite « tourniquet » ou encore « chacun son tour » (*round-robin*). C'est-à-dire, le canal de communication est utilisé par un dispositif émetteur à la fois. Cette technique procure une chance égale à tous les dispositifs de transmettre leurs données dans un intervalle de temps donné. Or, un dispositif peut prendre un temps exagéré pour transmettre ses données. Dans ce cas, la durée de l'intervalle de temps de la technique tourniquet peut varier d'un intervalle à l'autre. Pour plusieurs des applications industrielles, le déterminisme est de mise. Par exemple, un système contrôlant un bras manipulateur doit cadencer ses mouvements en fonction du déplacement d'un convoyeur. Les données indiquant le mouvement du convoyeur doivent être envoyées au contrôleur périodiquement dans un délai qui est constant. Pour y parvenir, la technique de multiplexage temporel peut être utilisée. Dans cette technique, les dispositifs émetteurs sont gérés par la technique tourniquet avec une contrainte supplémentaire qui consiste à limiter la durée d'exécution des dispositifs. Les émetteurs doivent transmettre leurs données dans le temps alloué, sinon ils doivent attendre leur tour suivant pour poursuivre la transmission. Le Tableau 2.2 présente quelques bus de terrain et le modèle de communication associé.

Tableau 2.2 Modèle de communication de trois (3) bus de terrain.

Bus	Type
Modbus	L'un des bus de terrain le plus ancien et répandu. Le modèle de communication est de type coordonnateur/nœud. C'est donc le coordonnateur qui réalise la technique de communication tourniquet.
CANBUS	Ce bus de terrain a été conçu pour des applications impliquant des microcontrôleurs. Le modèle de communication est de type diffusion de messages avec priorité. Tous les nœuds reliés au CANBUS reçoivent les messages et c'est au destinataire d'en faire la vérification pour la réception dans son nœud respectif. La priorité est à certains nœuds, leur permettant de transmettre et recevoir des messages en temps réel.

PROFIBUS	Ce bus de terrain utilise aussi un modèle de communication de type coordonnateur/nœud. Au lieu de réaliser uniquement la technique de communication tourniquet, le PROFIBUS permet aussi la communication acyclique où certains nœuds communiquent plus souvent que d'autres.

Les trois bus de terrain du Tableau 2.2 sont disponibles sous forme de nœuds dans la plateforme de développement Node-RED. En adoptant la plateforme de développement Node-RED, on récolte aussi une gamme de nœuds dédiés à l'interfaçage des bus de terrain.

2.4 Conclusions

En résumé, on a établi les différents choix quant aux technologies qui feront partie de la passerelle de communication. Comme plateforme de développement, ROS et Node-RED ont été sélectionnées pour faire partie du système. ROS sera utilisé pour programmer des fonctions et créer des nœuds qui opèreront un robot. Node-Red sera utilisé pour interfacer les bus de terrains. Ces deux plateformes auront donc des tâches fondamentalement différentes et pourront ainsi coexister dans un seul système. Quant au protocole de communication, MQTT a été retenu en raison de sa facilité d'intégration aux plateformes ainsi que du contexte d'utilisation de la passerelle de communication. Enfin, grâce au choix de Node-Red, différents bus de terrain pourront être intégrés à la passerelle au besoin.

CHAPITRE 3

CONCEPTION DU MODÈLE DE COMMUNICATION

3.1 Introduction

Jusqu'à présent, on a établi le contexte d'utilisation de la passerelle, fait une revue de littérature au sujet des différentes technologies qui pourraient être sélectionnées pour faire partie d'un concept de passerelle, puis sélectionné ces technologies. Dans cette section, les liens entre les différentes technologies sélectionnées seront établis et on présentera des architectures logicielles et matérielles. Une fois que les schémas auront été présentés, il y aura une section concernant les variations et les possibilités liées aux caractéristiques intrinsèques des technologies dans la passerelle.

3.2 Composants logiciels

Comme indiqué dans le chapitre 2, les trois technologies sélectionnées sont : i) ROS; ii) MQTT; iii) Node-RED. La Figure 3.1 est une représentation de ces technologies utilisées dans la passerelle de communication.

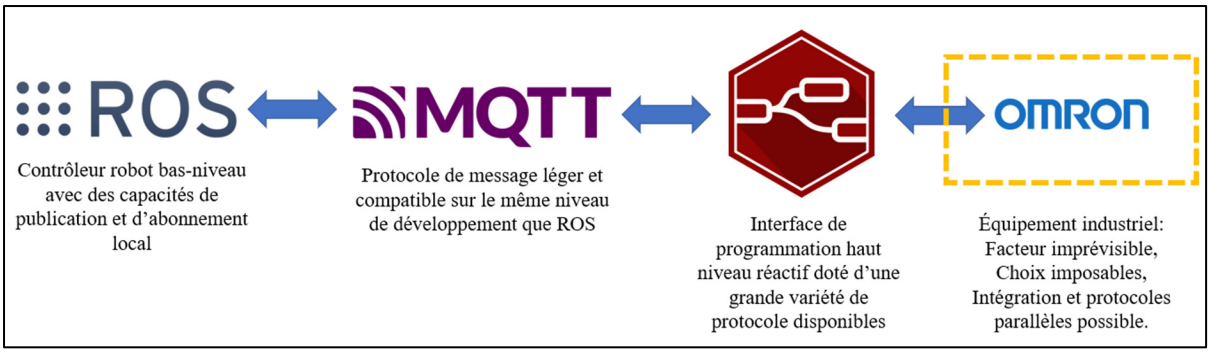

Figure 3.1 Technologies de la passerelle de communication

Dans la passerelle de communication, le système ROS est le premier système qu'il faut mettre en état de fonctionnement. En effet, tel qu'indiqué dans le premier objectif de ce projet de

recherche, l'utilité de ce système est de pouvoir s'interfacer à des robots à partir du code réutilisable. Idéalement, tous les composants désirés d'un système robotique seront contenus dans des packages ROS et le développement du système ne serait qu'un jeu d'assemblage. On atteindrait ainsi un taux de réutilisation du code maximal. En pratique, la disponibilité des packages ROS ne couvre pas tous les besoins d'un système robotique. Néanmoins, leur utilisation permet de réduire considérablement le travail de programmation.

Dans notre passerelle de communication, le système ROS interagira uniquement par le biais du protocole MQTT. En créant des instances clients MQTT dans le système ROS, on pourra faire transiter des topics MQTT avec ceux de ROS. En somme, lorsqu'un topic est défini dans ROS et que l'on désire l'envoyer au-delà de ROS, on créera alors un topic MQTT avec un client qui publiera l'information qui se trouve sur ROS. Dans la situation inverse ou l'on veut transmettre l'information dans le système ROS, des clients MQTT pourront alors aller s'abonner aux topics MQTT de systèmes intelligents dans l'environnement du robot. Le protocole MQTT fera également le pont avec la deuxième plateforme de développement, Node-RED. En effet, des nœuds réalisant ce protocole sont aussi disponibles dans Node-RED. En identifiant le broker MQTT, il est très facile d'ajouter des nœuds associés à des topics. Les objets industriels qui ne communiquent pas par le protocole MQTT et qui n'ont pas de package ROS seront pris en charge par le système Node RED à l'aide des réseaux de terrain. Enfin, les liens de la Figure 3.1 sont bidirectionnels et les technologies déployées peuvent transmettre et recevoir des données dans les deux directions.

3.2.1 Justification

Cet agencement de technologies est la recommandation de base pour l'utilisation de la passerelle de communication. Si les éléments de la passerelle ont déjà été justifiés dans le chapitre 2, leur jumelage aurait pu être différent. À la base, le système ROS a été sélectionné comme plateforme de développement pour ses fonctionnalités spécialisées en robotique. Idéalement, toutes les composantes utiliseraient un package pour être dans l'écosystème de

ROS, ce qui créerait des topics ROS. Bien entendu, ce n'est pas réaliste. Afin de ne pas alourdir son roulement interne de topics, on a fait le choix de prioriser son interfaçage avec MQTT afin de limiter la duplication des topics au travers de la collaboration de ROS, MQTT et Node-RED. Si plusieurs développeurs commençaient à utiliser la plateforme, ces bonnes pratiques les aideraient à savoir comment l'information se rend à la plateforme de développement et où l'on pourra consulter le registre de topic de ROS et MQTT.

Comme établi pour ROS, MQTT fera le pont avec Node-RED, la seconde plateforme de développement. Node-RED sera davantage utilisé pour l'interfaçage des bus de terrains. Ainsi, Node-RED sera davantage utilisé comme système de communication, et les développements seront concentrés sur ROS. Ainsi, avec ROS et MQTT, on pourra consulter l'ensemble des informations qui circulent sur le système.

Afin d'être efficace en termes d'intégration de protocoles, Node-RED sera la plateforme de développement utilisé pour interfacer les composantes utilisant des protocoles industriels. La diversité de nœuds disponibles pour les protocoles industriels ainsi que l'interface d'utilisateur conviviale sont les grands attraits de Node-RED. Ainsi, le temps de développement et d'intégration des différents protocoles sera grandement réduit par rapport au développement conventionnel. Ce dernier requiert une expertise en langage de programmation et des revues de code. L'arrivée imprévisible des différents bus de terrains sera donc plus facilement intégrable dans l'ensemble de la passerelle.

3.3 Composants matériels

Les matériels électroniques et électromécaniques peuvent varier d'une application à l'autre. Un exemple type est présenté dans la Figure 3.2.

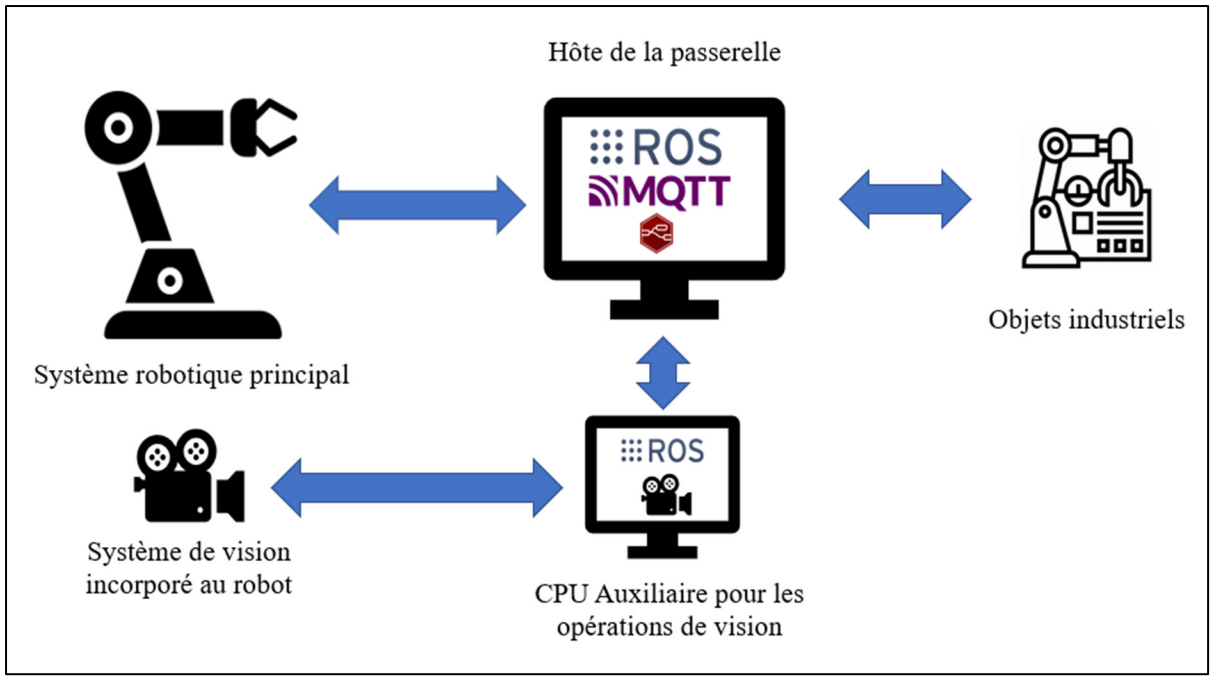

Figure 3.2 Composants matériels d'une application type

Il est possible de délocaliser des machines dans l'infrastructure de ROS en ayant un appareil qui agit en tant que ROS Master. Typiquement, le ROS Master est défini par défaut à l'adresse locale de l'hôte, mais il est possible de la délocaliser tant qu'il y a une adresse IP assignée et que le ROS Core est actif à cette adresse. Ce concept peut être intéressant si, par exemple, on veut faire des calculs lourds de trajectoires ou bien déterminer une cible à partir de l'acquisition d'un nuage de point. C'est le rôle qu'un ordinateur auxiliaire pourrait remplir. Les objets industriels dans cette architecture ne se limitent pas à ceux qui parlent uniquement des protocoles industriels : on peut inclure des appareils qui communiquent par MQTT ou qui ont même un ROS Package.

L'essentiel dans de la Figure 3.2 est de comprendre que la seule et unique composante qu'on ne devrait pas retrouver plusieurs fois dans le système est celle de l'hôte de la passerelle. Il est possible d'avoir plusieurs robots, des systèmes de vision, des appareils de calcul auxiliaire ou bien entendu plusieurs objets industriels intelligents dans l'infrastructure. Il reste cependant

essentiel de comprendre dans ce concept schématique de passerelle de communication qu'il y a un appareil central pour coordonner tous les messages. Cet appareil servira d'hôte au ROS Core, au serveur MQTT, ainsi qu'à contenir l'instance de Node.js et Node-Red. Jusqu'à présent, cet appareil doit avoir Ubuntu 18/20.04 pour avoir l'ensemble de ces systèmes, car le système ROS ne fonctionne à ce jour que sur le système Linux. Il est cependant possible d'utiliser une machine Windows pour être hôte de la passerelle à l'aide de la fonctionnalité WSL (*Windows Subsystem for Linux*).

En gros, WSL sert à exécuter un sous-système de Linux à même l'environnement de Windows. Ce sous-système doit toutefois utiliser une distribution d'Ubuntu. Ceci revient donc essentiellement à dire qu'il faut quand même Ubuntu pour pouvoir installer et utiliser ROS. Toutefois, cette solution est intéressante, car elle n'exige pas de matériel supplémentaire et il est même possible d'utiliser des librairies exclusives sur Windows à l'aide d'une connexion MQTT pour lier les informations à ROS. Une architecture logicielle sera d'ailleurs présentée dans la section suivante, qui exploite la passerelle à partir d'une machine Windows.

3.3.1 Justification

En tant que passerelle de communication pour un système robotique intelligent, il est très fréquent que des algorithmes très lourds en calculs soient déployés. Il est donc tout à fait possible qu'on ait besoin d'une machine auxiliaire pour ne pas surcharger un seul système. Une surcharge sur le processeur interne ou graphique d'un seul ordinateur pourrait nuire aux performances du système. Pire encore, le système pourrait simplement cesser de fonctionner. Cette architecture est une des nombreuses configurations possibles avec cet ensemble de logiciels pour la passerelle.

3.4 Conclusions

En conclusion, la passerelle de communication a été illustrée d'un point de vue logiciel puis matériel. Pour la structure logicielle, les bonnes pratiques et la logique derrière l'architecture logicielle basée sur les technologies sélectionnées dans le chapitre 2 ont été établies. En ce sens, les consignes associées sont plus rigides, car elles permettent un bon fonctionnement du système. Du côté matériel, on possède davantage de flexibilité puisque ce concept de passerelle peut être accueilli sur une variété de systèmes.

CHAPITRE 4

PROTOCOLE D'EXPÉRIMENTATION ET ANALYSE DES RÉSULTATS

4.1 Introduction

Dans ce chapitre, on établira le protocole d'expérimentation qui permettra d'évaluer la latence et le débit des données de la passerelle de communication conçue. Ces deux métriques permettront de mesurer quantitativement les performances du système. La latence est une métrique essentielle à mesurer dans notre système puisqu'elle viendra indiquer s'il est possible de faire des applications en temps réel avec le système robotique. Quant au débit de données, cette métrique permettra d'avoir une meilleure idée de la capacité maximale du système. À ce stade, on ne sait pas encore à quelle fréquence on pourra opérer nos procédés et on ne connaît pas non plus la capacité maximale du système pour accueillir d'autres sous-systèmes matériels. L'équipement utilisé dans l'évaluation de performance comprend un ordinateur et un automate programmable reliés en réseau. La configuration de ces équipements sera présentée dans la section 4.2.

4.2 Configurations des équipements

On a utilisé un ordinateur Windows et son sous-système de WSL (*Windows Subsystem for Linux*) pour réaliser la passerelle de communication. Comme indiqué dans la section 3.3 du chapitre 3, le WSL est nécessaire, car le système ROS n'est présentement pas disponible sur Windows. Par le fait même, le serveur MQTT se trouve également dans l'environnement WSL puisqu'il n'y pas a d'environnement de programmation sur le système d'opération de Windows établi et que le système est plus léger sur Linux. Ce dernier est relié en réseau avec l'automate Omron NX102. La Figure 4.1 est un schéma montrant les relations logiques de la passerelle de communication, l'ordinateur hôte et l'automate programmable. Ces équipements forment le banc d'essai pour l'exécution des expériences.

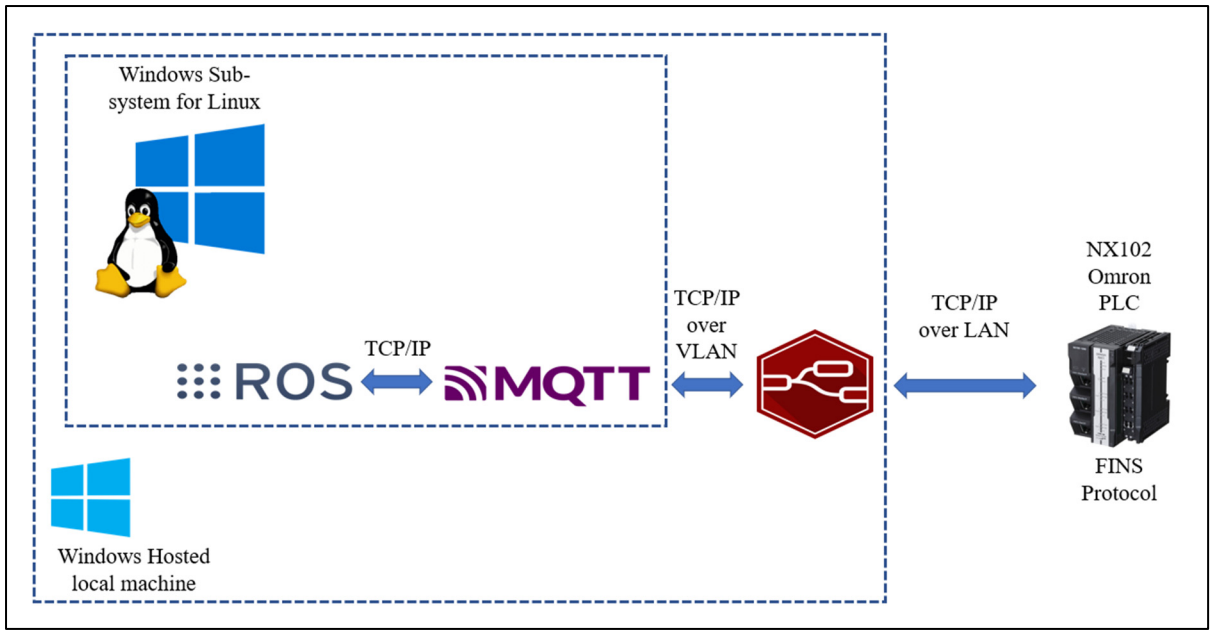

Figure 4.1 Banc d'essai pour la mesure des métriques de performance

Dans ce banc d'essai, l'automate NX102 joue le rôle d'objet industriel connecté. Le NX102 est un modèle d'automate couramment utilisé dans l'automatisation industrielle. Ce choix repose principalement sur sa compatibilité avec un nombre de réseaux de terrain et sa petite taille. Puisque le NX102 est un automate industriel, sa sélection rendra les résultats de ce banc d'essai plus réalistes en termes d'opération et en termes de qualité.

4.2.1 Ordinateur hôte

La configuration de l'ordinateur hôte est présentée dans le Tableau 4.1. Il s'agit d'un ordinateur portable de marque Lenovo sans aucune caractéristique spéciale. Il est à noter que la taille des antémémoires peut influencer la mesure des métriques de performance. En effet, lors de l'exécution du banc d'essai, l'ordinateur hôte doit invoquer le sous-système WSL de Windows. Dans la version 2 de WSL, le sous-système Linux s'exécute dans une machine virtuelle. La taille des antémémoires de l'ordinateur hôte peut accélérer ou ralentir l'exécution de cette

machine virtuelle. Cependant, la présence de la carte graphique NVIDIA n'est pas nécessaire et n'apporte aucun avantage dans la mesure des métriques de performance.

Tableau 4.1 Spécifications techniques de l'ordinateur hôte

Processeur	Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
Antémémoires	256KB (L1), 1MB (L2), 8MB (L3)
Mémoire vive	24 GB
Système d'opération	Windows 10 Pro
Numéro de version	21H2 19044.2728
Type de système	64 bits
GPU	NVIDIA Quadro P520
Carte réseau	TAP-Windows Adapter V9

4.2.2 Windows 10 et WSL

Le système ROS ne peut s'exécuter sur un ordinateur Windows. On peut contourner ce problème à l'aide du sous-système WSL de Windows. La raison pour laquelle le système Windows est maintenu dans le banc d'essai est fort simple : la plupart des applications en automatisation industrielle sont conçues pour le système d'exploitation Windows et non pas le système Linux. C'est le cas pour les automates programmables et c'est particulièrement vrai pour l'automate NX102 d'Omron.

En ce qui concerne le sous-système WSL, son application est intéressante, car il permet aux logiciels Windows et Linux d'être exécutés dans le même ordinateur hôte d'une façon transparente. On peut ainsi utiliser les applications de l'automate NX102 tout en exécutant la passerelle de communication sous le système ROS. Grâce au sous-système WSL, on peut utiliser deux systèmes d'exploitation en même temps sur le même ordinateur. WSL est une fonctionnalité de base dans Windows 10. En utilisant une ligne de commande dans un terminal Powershell, on peut alors activer cette fonctionnalité. Une fois que c'est fait, il suffit d'installer

la version désirée de Linux distribution Ubuntu à partir du magasin d'applications de Microsoft (*Microsoft Store*).

Le WSL ajoute une couche de virtualisation dans l'utilisation de la passerelle de communication. Cette couche supplémentaire dans le traitement des informations à l'interne au travers du processeur peut potentiellement modifier les performances du système. En effet, des tests sur différentes applications dans un système intrinsèquement Windows ont été réalisé pour observer l'influence d'activer WSL ou non pendant leur usage. Les résultats démontrent qu'elle peut occasionner des ralentissements de 2% pour certaines applications. (Kinghorn, 2020) Cette influence sur les résultats sera comprise afin d'évaluer adéquatement les performances de la passerelle sur une machine avec Windows 10.

4.2.3 Système ROS

WSL utilise la distribution d'Ubuntu 20.04 LTS. Par conséquent, la version de ROS adéquat est celui nommé Noetic. Son installation a été faite en utilisant la procédure standard établie par le site officiel de ROS pour obtenir la version à jour. À partir de son interface, plusieurs nœuds ROS sont démarrés parallèlement afin d'exécuter un envoi et une réception de données à partir de ROS. Le Tableau 4.2 résume les différents nœuds ROS. La chaîne d'évènement sera présentée dans la section 4.3.1.

Tableau 4.2 Configuration des nœuds ROS

Noeud	Fréquence	Rôle
Node_Publisher	300Hz	Publie la valeur initiale et affiche l'horodatage
MQTT_Sender	Réactif	Reçoit le topic ROS et l'envoie sur MQTT
MQTT_Receiver	Réactif	Reçoit le Topic MQTT et le publie sur un topic ROS
Node_Subscriber	Réactif	Reçoit le Topic ROS et donne l'horodatage

4.2.4 Broker MQTT

Afin de pouvoir utiliser le protocole MQTT, il faut avoir un serveur MQTT qui roulera dans la machine hôte de la passerelle de communication. Pour cette expérimentation, le serveur Mosquitto MQTT d'Eclipse Mosquitto a été utilisé. Mosquitto est un serveur de MQTT très populaire et peu demandant sur le système, ayant un poids léger sur l'infrastructure (Light, 2017). En utilisant la commande APT sur Linux, il ne suffit que l'appel d'une seule commande pour exécuter son installation. Par la suite, il faut alors activer le serveur pour que tous les clients établissent dans le système ROS et Node-Red fonctionne. Le Tableau 4.3 donne la configuration des quatre instances de clients MQTT (dans les systèmes ROS et Node-RED).

Tableau 4.3 Configuration des clients MQTT

Nom du client	Adresse	port	Topic	QoS	Keep_alive
ROS_Publisher	WSL_IP	1883	Chatter	0	60
Node_Subscriber	WSL_IP	1883	Chatter	0	60
Node_Publisher	WSL_IP	1883	Feedback	0	60
ROS_Subscriber	WSL_IP	1883	Feedback	0	60

4.2.5 Node.js et Node-RED

Node-Red a été installé nativement sur la machine Windows pour pouvoir adresser directement l'adresse externe de l'automate programmable NX102. Afin de pouvoir installer Node-Red, il faut auparavant avoir une installation de Node.js. En utilisant l'installateur officiel de Node.js sur son site, il faut utiliser la commande NPM (*Node Package Manager*) pour installer l'instance de Node-Red sur Windows. Une fois que l'installation est faite, il faut alors créer un écoulement qui créera le lien entre MQTT et l'automate programmable. Dans le flux utilisé, il y a deux sous-flux qui sont utilisés. Le Tableau 4.4 résume ces flux et leurs fréquences.

Tableau 4.4 Configuration de Node-RED

Flux	Fréquence	Rôle

Réception	Réactif	Reçoit le Topic MQTT, Publie dans FINS
Renvoi	300Hz	Reçoit de FINS et Publie sur MQTT

4.2.6 Automate Omron et protocole FINS

Le protocole natif de communication utilisé pour échanger avec l'automate programmable NX102 est le FINS. Afin de pouvoir être utilisé, il faut impérativement faire plusieurs étapes. Comme mentionné dans la revue de littérature, FINS est un protocole qui fonctionne à partir d'adressage de variables dans des cases mémoire. Afin d'ouvrir la plage de données, il faut activer la plage nommée DM (*Data Memory*) dans la configuration de mémoire du PLC à partir du logiciel Sysmac. La plage de données DM est une plage de donnée qui ouvre une banque d'adresse utilisable par le protocole FINS. Une fois que cette plage est disponible, on peut publier jusqu'à 32768 entiers sur FINS avec cette plage de données. La Figure 4.2 illustre le menu de ces données dans Sysmac.

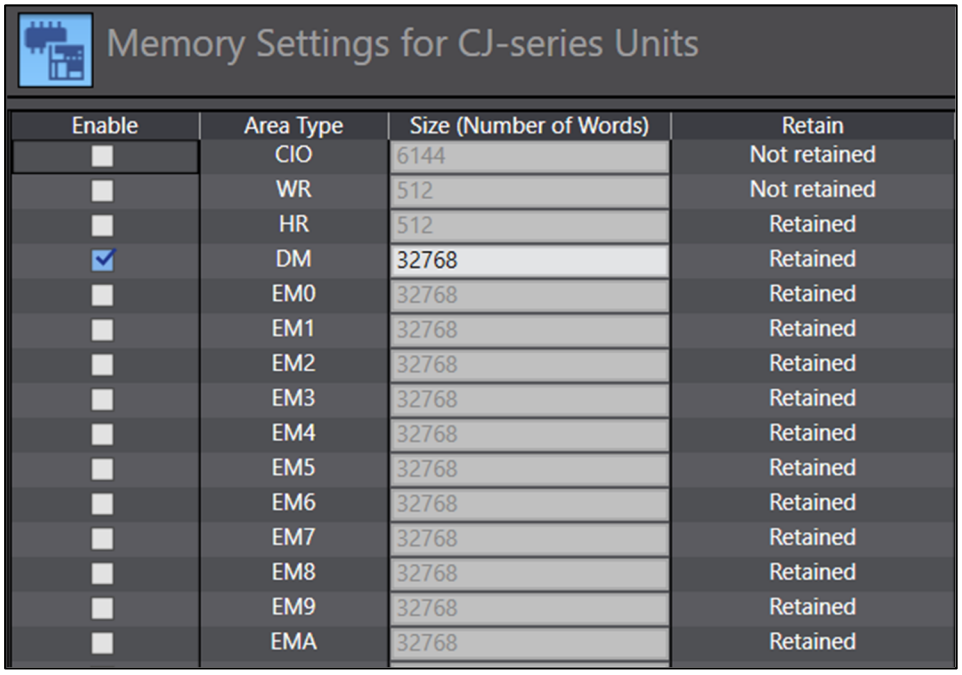

Figure 4.2 Configuration des paramètres mémoires dans Sysmac

Une fois que cette plage est activée, deux variables de type entier ont été déclarées et adressées aux cases mémoire %D100 et %D200. Ces variables sont internes dans le système. Une fois que ces variables ont été déclarées, une ligne de commande dans un programme de schéma à contacts (*Ladder Programming*) est déclarée afin d'acheminer la variable reçue (%D100) à la variable envoyée (%D200). Afin de ne pas simplement copier la variable, une opération mathématique a été réalisée pour rendre l'entier négatif, comme démontré dans la Figure 4.3. Afin de pouvoir facilement distinguer l'entier aléatoire publié par ROS, la transformation la plus simple à faire était de rendre le chiffre négatif. Ainsi, en voyant que la valeur est négative à son retour dans ROS, on s'assure que la variable envoyée est bel et bien passée par l'automate programmable.

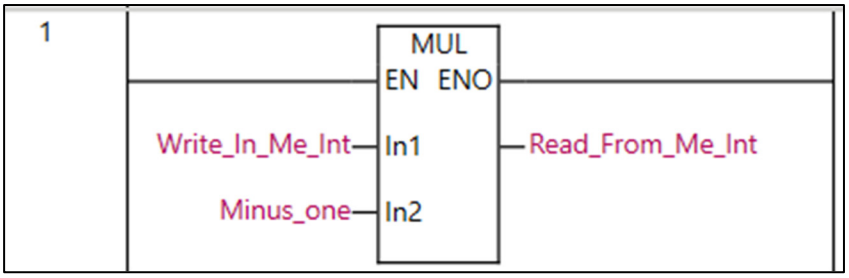

Figure 4.3 Programme dans Sysmac

Comme indiqué dans la Figure 4.3, il n'y a pas de condition à ce que l'étape se mette à jour. La transformation se réalise donc à chaque cycle de balayage de l'automate. Ce temps de cycle n'a pas pu être mesuré, car ce dernier varie en fonction des calculs sur l'automate. Toutefois, étant donné que ce modèle est spécialisé pour des applications en temps réel (Omron, 2023), le temps de cycle dépasse largement la vitesse de 300Hz, ce qui ne créera pas un goulot dans la chaîne. En somme, la valeur retrouvée à l'adresse %D100 (`Write_In_me_Int`) se fait multiplier par -1 et est inscrite dans %D200 (`Read_From_Me_Int`). Dans ce cas ici, l'automate est en mode strictement réactif.

4.3 Expérimentation

Afin de pouvoir mesurer la latence du système de transmission, la plus grande chaîne de sous-systèmes a été utilisée, c'est-à-dire, du système ROS jusqu'à l'automate puis de retour dans le système ROS. Dans ce cas, la métrique aller-retour pour la latence RTL (*Roundtrip latency*) peut être appliquée. Le principe de mesure est basé sur la figure 4.4 ci-dessous.

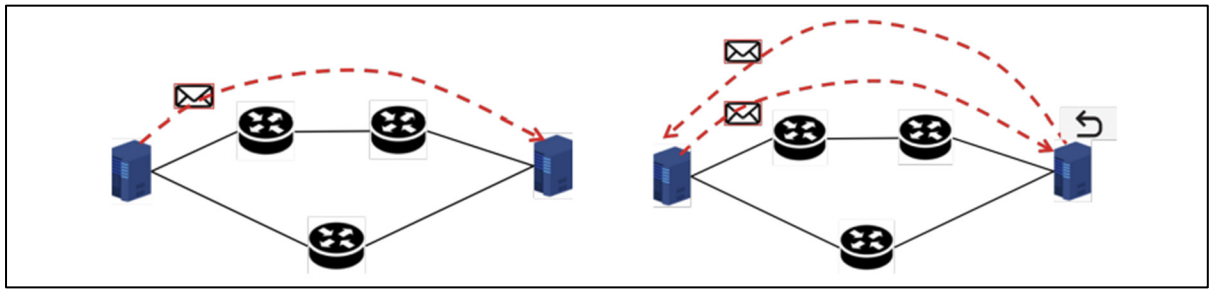

Figure 4.4 Mesure de latence à sens unique et à aller-retour
Tirée de Sundaresan et al. (2020)

En utilisant cette méthodologie, on a pu se fier sur le système de temps dans ROS basé sur l'horloge en temps réel de Linux. L'envoi et la réception des données sont alors indiqués puis notés à partir du terminal qui affiche l'envoi et la réception de messages avec un horodatage. En mesurant le delta entre l'horodatage d'envoi et de réception du message, on pourra observer la quantité de temps utilisée pour qu'un message soit mis à jour sur l'ensemble du système. En ce qui concerne le débit d'un message, on pourra alors mesurer la quantité de messages qui ont été reçus en une seconde. La méthode d'envoi et le paramétrage du système seront élaborés dans la section programmation et schéma.

4.3.1 Prise de mesure

Afin de résumer la programmation derrière l'envoi de télémétrie en aller-retour pour la mesure de la latence et du débit, la Figure 4.5 expose l'ensemble des opérations réalisées dans la passerelle de communication.

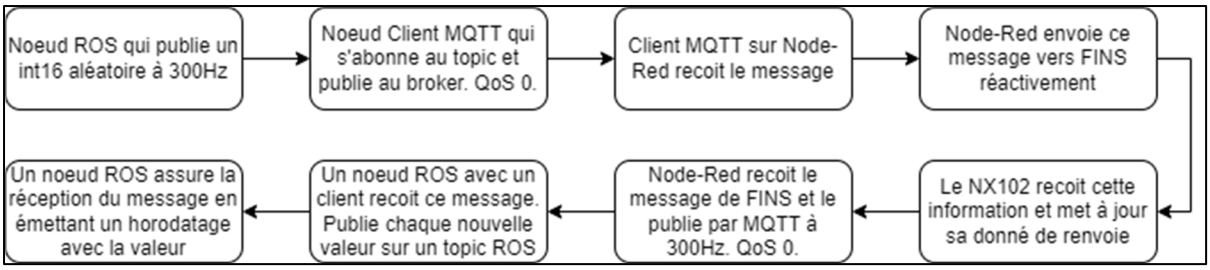

Figure 4.5 Schéma d'évènement

En somme, un nœud ROS publiera dans un topic utilisant une structure de message standard Int16, qui équivaut à un entier de 16 bits, et est publié à une fréquence de 300Hz avec un horodatage. Ce topic est alors lu par un autre nœud ROS possédant un client MQTT qui publiera à son tour un topic sur MQTT. Un client dans Node-Red s'abonne à ce client et publie réactivement la donnée dans la case mémoire %D100 dans FINS, à l'aide d'un nœud FINS Write dans Node-Red, qui sert à publier des données par FINS. Cette donnée est alors reçue dans la case mémoire du PLC et met à jour la valeur de la case mémoire %D200 avec la valeur négative de %D100. Cette valeur fait ensuite demi-tour avec un nœud FINS Read qui lie la case mémoire %D200. Cette valeur est publiée à une fréquence de 300Hz sur un topic MQTT. Un autre nœud ROS est abonné au topic de retour et publie alors en réaction exclusivement les nouvelles valeurs qu'elle reçoit. Enfin, ce topic ROS est affiché avec un horodatage. Le Tableau 4.5 résume l'ensemble des paramètres de configuration de la chaîne.

Tableau 4.5 Résumé des paramètres

Format de donnée	Fréquence de publication lorsqu'applicable	QoS	Keep_Alive
int16 (entier16 bits)	300Hz	0	60

Cette expérience a été menée pendant une centaine d'instances pour mesurer le débit de données en une seconde ainsi que la latence d'aller-retour. Afin de rendre cette expérience représentative de la réalité, on a sélectionné le format de donnée le plus couramment utilisé dans l'automatisation des procédés, soit une variable entière signée à 16 bits. Des fréquences

de publications de 300Hz ont été utilisées pour maximiser l'envoi de message à chaque nœud qui n'est pas réactif afin de ne pas créer de latence artificielle dans le système. Enfin, des expérimentations parallèles ont été menées avec différentes structures de message, dont une variable booléenne et un entier à 8 bits. Toutefois, les résultats observés n'ont pas varié par rapport aux performances du système avec un entier à 16 bits, donc ces résultats n'ont pas été jugés pertinents à explorer davantage.

4.3.2 Justification

Les deux métriques seront mesurées sont la latence RTL ainsi que le débit. Le débit peut être mesuré par unité de temps. On a choisi de le mesurer sur une période d'une seconde. Le choix de ses deux métriques a été fait afin de déterminer les limites du système. La latence a été choisi comme métrique afin de vérifier le seuil d'application en temps réel de la passerelle. Ainsi, on pourra voir à quelle fréquence le système sera en mesure de se synchroniser à différents sous-systèmes hors du contrôle de la passerelle. Quant au débit, cette métrique permettra d'établir les limites de capacité du système. De plus, le débit nous indiquera s'il est nécessaire d'effectuer des envois de messages en parallèle afin d'optimiser le système. On aura alors une meilleure idée sur les capacités de notre passerelle de communication.

4.4 Analyse statistique des résultats

Des mesures ont été prélevées en appliquant la procédure de prise de mesure de la section 4.3.1. Dans son état brut, ces mesures ne présentent que des statistiques de l'échantillon prélevé. Ce qui est intéressant est une estimation des statistiques qui sont représentatives de la population des mesures (l'ensemble de toutes des mesures possibles). Dans l'expérimentation réalisée, on a mesuré 100 valeurs de latence et de débit de données. Afin d'estimer les statistiques de la population, la méthode de Bootstrap est utilisée (Efron et al., 1985). Cette dernière est une méthode statistique non paramétrique reposant sur la construction d'une nouvelle distribution à partir des mesures prélevées à l'aide de l'échantillonnage avec remise. Les avantages de cette méthode statistique sont :

- Étant non paramétrique, le Bootstrap ne suppose pas une distribution gaussienne des mesures. Dans notre étude, les valeurs de RTL et du débit sont probablement très proches les unes des autres avec une répartition qui n'est pas symétrique autour de la valeur moyenne;
- Aucune donnée supplémentaire n'est nécessaire pour son application. Le Bootstrap utilise seulement les mesures échantillonnées pour réaliser l'estimation des statistiques.

La Figure 4.6 illustre le principe de fonctionnement de la méthode Bootstrap, tirée de Greenwell et al. dans leur ouvrage sur le traitement des données.

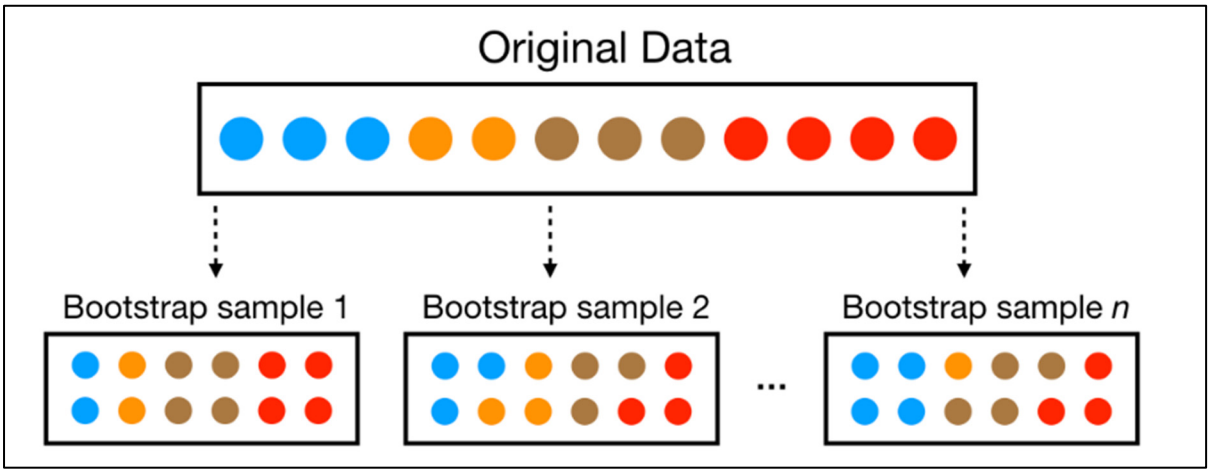

Figure 4.6 Processus de la méthode Bootstrap
Tirée de Greenwell et al. (2023)

Cette méthode permettra alors de créer d'autres données qui seront alors analysées pour ressortir d'autres paramètres. La bibliothèque Pandas de Python a été utilisée pour produire ces données. Un total de $B = 1000$ rééchantillonnages ont été effectués par Bootstrap sur les deux ensembles de $N = 100$ mesures (RTL et débit des données). À partir de ces échantillons, on peut y calculer différents paramètres. Ultimement, on y obtiendra une distribution normale des échantillons. Les Figures 4.7 et 4.8 démontrent les distributions normales obtenues à partir de ces échantillonnages.

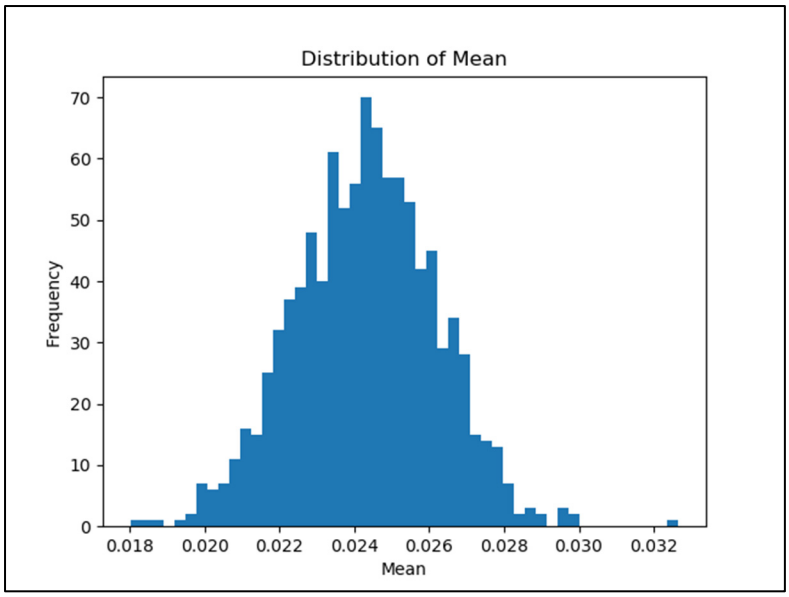

Figure 4.7 Distribution des moyennes
des échantillons du RTL

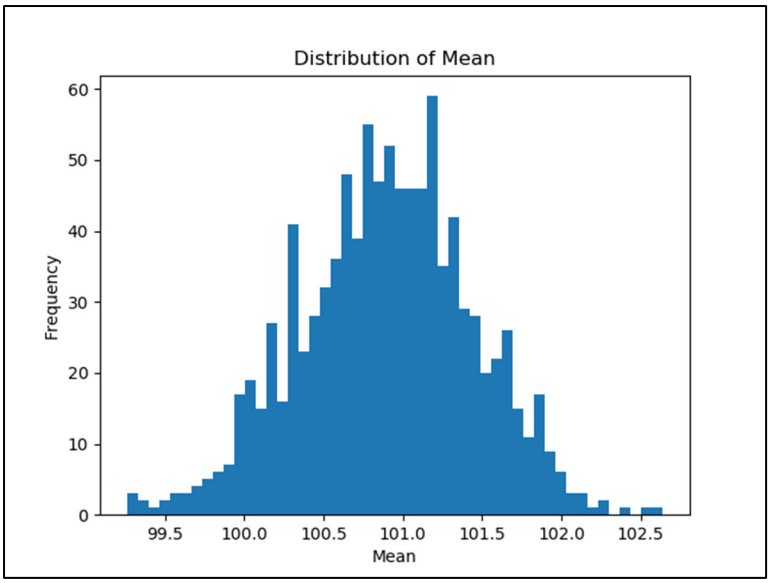

Figure 4.8 Distribution des moyennes des
échantillons du débit

Enfin, plusieurs statistiques ont été calculées pour faciliter l'analyse des résultats. La première statistique est celle de la moyenne et elle est présentée dans l'équation 4.1 :

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} \tag{4.1}$$

où N est le nombre de mesures dans l'échantillon, x_i sont les mesures et $\bar{x}$ est la moyenne obtenue à l'échantillon. La méthode Bootstrap réalise B rééchantillonnages avec remise et la moyenne statistique est la valeur moyenne de ces B rééchantillonnage. En plaçant les mesures x_i en ordre croissant, les rangs de la valeur sont déterminés puis la médiane est alors calculée de la façon suivante :

$$m = \begin{cases} x_{\frac{N+1}{2}}, & \text{si } N \text{ est impair,} \\ \dfrac{x_{\frac{N}{2}} + x_{\frac{N}{2}+1}}{2}, & \text{si } N \text{ est pair.} \end{cases} \tag{4.2}$$

Dans (4.2), $x_{\frac{N}{2}}$ signifie la mesure à la position $N/2$ de l'échantillon. Encore une fois, la médiane Bootstrap est la moyenne des médianes de B rééchantillonnages. La variance et l'écart-type des mesures représentent la dispersion des mesures autour de la valeur moyenne. Pour la variance s^2, elle est calculée par

$$s^2 = \frac{\sum_{i=1}^{N}(x_i - \bar{\mathrm{x}})^2}{N-1} \tag{4.3}$$

et l'écart-type σ est alors simplement

$$\sigma = \sqrt{s^2}. \tag{4.4}$$

Le coefficient de l'asymétrie u (*skewness*)

$$u = \frac{\sum_{i=1}^{N}(x_i - \bar{\mathrm{x}})^3}{(N-1) \times \sigma^3} \tag{4.5}$$

donne une indication sur l'asymétrie de la distribution des mesures prélevées. Une valeur positive du coefficient signifie que la distribution est décalée vers la gauche de la médiane. Une valeur négative signifie que la distribution est décalée vers la droite de la médiane. La distribution normale possède un coefficient de l'asymétrie qui est nul. Enfin, on peut aussi estimer le coefficient d'aplatissement k (*kurtosis*) de la distribution des mesures par

$$k = \frac{\sum_{i=1}^{N}(x_i - \bar{\mathrm{x}})^4}{N \times \sigma^4}. \tag{4.6}$$

Encore une fois, les statistiques (4.1) à (4.6) ont été calculés à partir de B = 1000 rééchantillonnages avec remise de la méthode Bootstrap. Enfin, il y aura un intervalle de

confiance pour chaque paramètre. L'équation 4.7 indique comment obtenir les bornes de l'intervalle avec un taux de 95%. La variable ci représente les deux valeurs. La variable $\bar{x}$ indique la moyenne, σ l'écart-type et *n* la taille de l'échantillon d'où

$$ci = \bar{x} \pm 0.95 \frac{\sigma}{\sqrt{n}}. \quad (4.7)$$

Les intervalles de confiance seront utilisés pour chaque paramètre afin d'établir la plage des valeurs statistiquement significatives. Puisqu'on travaille avec des échantillonnages, il y a une nature aléatoire à notre échantillonnage. En établissant un paramètre de confiance de 95%, on peut voir les variabilités possibles à nos résultats si on reproduisait l'échantillonnage avec la méthode Bootstrap.

Maintenant que les formules pour les différents paramètres et leur manipulation ont été établies, les Tableaux 4.6 et 4.7 représentent respectivement le cumul des métriques obtenues à partir des résultats de l'expérimentation, qui servait à mesurer la latence et le débit de la passerelle de communication. La latence a été mesurée en secondes (s) et le débit en message transmis par seconde (msgs/s).

4.4.1 Latence RTL

Pour la latence moyenne observée à partir de l'expérience, on a obtenu une moyenne de 24 millisecondes, avec des limites de 21 à 28 millisecondes pour intervalle de confiance de 95%. En moyenne, pour une transmission de donnée FINS à partir d'un capteur vers son contrôleur, la latence est d'environ 7 millisecondes (Ramadhan et al., 2013). Par rapport à notre système qui passe par plusieurs couches de protocoles pour faire un aller-retour dans le système de communication, une latence de 24 millisecondes semble adéquate. Pour une application de robotique collaborative, ce délai sera acceptable dans des scénarios ou le robot n'a normalement pas à se synchroniser avec un objet industriel dans son environnement.

La valeur moyenne de la médiane est de 13 millisecondes, avec un intervalle de confiance à 95% de 9 à 21 millisecondes. Le fait que l'intervalle de confiance de la médiane soit inférieur à l'intervalle de la moyenne indique qu'il y a la présence de valeurs éloignées par rapport à la moyenne. La distribution de données n'est donc pas symétrique par rapport à la médiane.
La variance et l'écart-type montrent que les valeurs de latence sont peu dispersées autour de la valeur moyenne. Cette affirmation est motivée par le coefficient de variation cv = s /m où s est l'écart-type et m est la valeur moyenne. Prenons l'écart-type et la moyenne des échantillonnages et on obtient un coefficient de variation de 0.7917. Le coefficient de variation étant inférieure à 1, on peut considérer que les valeurs mesurées de latence sont peu dispersées.

La dispersion est reliée à la précision (à quel point les mesures de latence sont proches les unes des autres). Ainsi, plus la dispersion est petite, plus la précision est grande. On peut donc accorder une grande confiance aux mesures obtenues.

L'asymétrie de la distribution des valeurs de latence est illustrée par la différence entre la latence moyenne et médiane. La valeur moyenne est de 0.75 et l'intervalle est de 0,35 à 1,2. Ceci montre que la distribution est asymétrique vers la droite, avec des valeurs positives indiquant que les valeurs de latence sont plus susceptibles d'être en dessous de la moyenne. Cependant, il est important de noter que la valeur de l'asymétrie d'une distribution normale est nulle. Avec une valeur positive généralement entre 0.5 et 1, la règle du pouce indique que les pointes dans les mesures sont généralement plus rapides que la valeur moyenne.

L'aplatissement est une mesure des valeurs aberrantes. Ces valeurs aberrantes se trouvent dans les extrémités d'une distribution. Pour une distribution normale, la valeur de l'aplatissement est de 3. Avec un aplatissement moyen de 1.9 et un intervalle de confiance de 1.5 à 2.8, on peut croire qu'on a une moins grande quantité de données aberrantes dans les 100 mesures de latence effectuées.

Le Tableau 4.6 résume l'ensemble des résultats obtenus de cette analyse.

Tableau 4.6 Résumé des paramètres et intervalles de confiance de la latence

Valeur minimale	0.0086s	
Valeur maximale	0.0669s	
Moyenne des métriques à partir des échantillons	Limite inférieure de l'intervalle de confiance	Limite supérieure de l'intervalle de confiance
Moyenne des échantillons		
0.024s	0.021s	0.028s
Médiane des échantillons		
0.013s	0.0092s	0.021s
Variance des échantillons		
0.00036s	0.00029s	0.00044s
Écart-type des échantillons		
0.019s	0.017s	0.021s
Coefficient de l'asymétrie des échantillons		
0.75	0.35	1.2
Coefficient de l'aplatissement des échantillons		
1.9	1.5	2.8

4.4.2 Débits des données

Pour le débit moyen du système, on a obtenu une moyenne de 100 messages par secondes avec un intervalle de confiance de 99.8 à 102 messages. Avec un débit de 100 messages par seconde, on peut raisonnablement bien transmettre plusieurs variables parallèlement. Ceci peut être expliqué par la limitation d'un sous-système, plus spécifiquement celui de Node-Red.

Normalement, MQTT et ROS peuvent largement transmettre des messages à une fréquence de 300Hz, tel qu'exécuté dans le système.

La médiane obtenue est de 100.52 messages par seconde avec un intervalle de confiance de 100 à 102. Étant donné que cet intervalle est pratiquement superposé avec celui de la moyenne, on peut s'attendre à ce que la distribution des données soit très symétrique.

La variance et l'écart-type montrent que les valeurs de latence sont peu dispersées autour de la valeur moyenne. À nouveau, comme pour le phénomène observé dans les mesures du débit, notre coefficient de variation (écart-type divisé par la moyenne) est de 0.0565. Avec un coefficient de variation si proche de zéro, les mesures de débit sont très peu dispersées. Les mesures sont donc très constantes. On peut donc accorder encore une fois une grande confiance à ces mesures.

Pour le coefficient d'asymétrie, on a obtenu une valeur moyenne de -0.0332, avec un intervalle de confiance de -0.835 à 0.86. Avec une telle valeur, on peut s'attendre à ce que la distribution soit en effet très symétrique. À l'aide de la règle du pouce, l'intervalle de confiance peut laisser croire que la distribution est modérément symétrique, mais les facteurs précédents et la valeur moyenne indiquent que la distribution est fort probablement symétrique.
Enfin, on a un coefficient d'aplatissement de 1.611, avec un intervalle de confiance de -0.433 à 3.173. Comme établi précédemment, une distribution normale aura un coefficient inférieur à 3. L'ensemble de l'intervalle de confiance se trouve pratiquement sous la barre de 3, avec une moyenne de 1.611. On peut donc croire qu'on a une distribution normale des données et donc peu de données aberrantes.

Le Tableau 4.7 résume l'ensemble des résultats obtenus de cette analyse.

Tableau 4.7 Résumé des paramètres et intervalles de confiance du débit

Valeur minimale		83 msgs/s
Valeur maximale		121 msgs/s
Moyenne des métriques à partir des échantillons	Limite inférieure de l'intervalle de confiance	Limite supérieure de l'intervalle de confiance
Moyenne des échantillons		
100.91 msgs/s	99.81 msgs/s	101.98 msgs/s
Médiane des échantillons		
100.52 msgs/s	100 msgs/s	102 msgs/s
Variance des échantillons		
32.19 msgs/s	21.26 msgs/s	44.79 msgs/s
Écart-type des échantillons		
5.65 msgs/s	4.611 msgs/s	6.69 msgs/s
Coefficient de l'asymétrie des échantillons		
-0.0332	-0.835	0.86
Coefficient de l'aplatissement des échantillons		
1.611	-0.433	3.173

4.5 Conclusions

Pour conclure, dans cette section, on retrouve tous les éléments nécessaires afin de reproduire l'expérience menée pour évaluer la passerelle de communication. À partir de la mise en œuvre

du système, on y a mesuré la latence RTL ainsi que le débit de l'envoi d'entiers au travers d'une chaîne d'évènements dans la passerelle de communication. Avec les résultats récoltés, on a effectué une analyse statistique permettant de pouvoir affirmer avec confiance que les résultats obtenus sont fiables et représentatifs des performances du système.

CONCLUSION ET RECOMMANDATIONS

L'objet principal de ce projet de recherche est de concevoir une plateforme logicielle pouvant communiquer avec une gamme de robots industriels et des dispositifs connectés. Nous avons retenu les systèmes ROS, Node-RED et le protocole MQTT dans la conception de la solution logicielle. L'intégration judicieuse de ces technologies permet de bien combler les besoins exprimés et de réaliser la passerelle de communication souhaitée.

Le système ROS est un puissant cadriciel permettant le développement de nouvelles fonctionnalités et est compatible avec une large gamme de robots industriels. De plus, ROS est un cadriciel à source ouverte soutenu par une grande communauté de développeurs. L'évolution de ce système est donc assurée à court et à moyen terme.

Le second objectif formulé était de développer une infrastructure qui servira de passerelle de communication intégrant une multitude d'interfaces qui seront interconnectées. La passerelle de communication a été conceptualisée en tenant compte des contraintes de l'objectif 1. En effectuant une sélection parmi différentes technologies candidates, on a créé une infrastructure interfaçant ROS, MQTT, Node-RED ainsi qu'un nombre de dispositifs industriels. Cette passerelle est en mesure de transmettre et de recevoir des données d'un robot et des dispositifs connectés à des réseaux de terrain.

Le troisième objectif a été atteint par la quantification des temps de latence dans la passerelle de communication conçue. Pour mesurer les temps de latence, nous avons effectué des tests à partir d'un protocole expérimental et défini la composition matérielle pour obtenir des résultats réalistes. En utilisant les métriques qui sont la latence du système et le débit des données, nous avons pu valider la performance du système avec une méthode statistique non paramétrique. Ces résultats ont montré que notre système était capable de communiquer de manière efficace et en temps réel.

Un point important est à retenir. Le version ROS utilisé dans ce projet de recherche est la version appelée Noetic. Son support à long terme (Long-term support) prendra fin en 2025. Il faut donc prévoir la migration du système ROS vers une version future. Le système ROS ne possède que très peu de capacité en matière de sécurité informatique et de sécurité machine. Il est donc recommandé de poursuivre le projet de recherche en mettant l'accent sur les aspects touchant à la sécurité physique et logicielle de la solution. Un autre point à considérer est relié à la mise en œuvre éventuelle de la passerelle de communication. Les tests et les mesures effectués ont été réalisés dans un environnement contrôlé. Le déploiement de la passerelle sur le terrain aura certainement des impacts sur la performance de celle-ci. C'est pourquoi il serait intéressant de refaire les mesures de performance sur les lieux de déploiement et d'analyser de nouveau les résultats en utilisant la même procédure statistique présentée dans ce mémoire. Aussi, certains paramètres d'expérimentation peuvent être modifiés. Par exemple, utiliser un ordinateur hôte d'architecture différente tel que le ARMv8 d'un Jetson, l'architecture AMD des systèmes Windows ou encore l'architecture AArch64 des ordinateurs Apple. On pourrait aussi utiliser un ordinateur hôte muni d'un système d'exploitation Ubuntu et appliquer une machine virtuelle pour exécuter des applications Windows. On pourra alors comparer les performances obtenues avec celles de ce projet de recherche et identifier la meilleure configuration de l'ordinateur hôte.

Enfin, la conception de la passerelle de communication est le résultat d'une exploration de technologies liée à l'utilisation d'un robot dans l'aide technique des personnes en perte de mobilité. La solution logicielle proposée dans ce mémoire en est un élément central. Elle est une assise sur laquelle d'autres éléments de l'application robotique viendront apporter leur contribution.

BIBLIOGRAPHIE

Ahmed, Mohiuddin, Adnan Anwar, Abdun Naser Mahmood, Zubair Shah et Michael J. Maher. 2015. « An Investigation of Performance Analysis of Anomaly Detection Techniques for Big Data in SCADA Systems ». EAI Endorsed Transactions on Industrial Networks and Intelligent Systems, vol. 2, no 3, p. e5-e5.

Alani, Mohammed M. 2014. Guide to OSI and TCP/IP Models. Coll. « SpringerBriefs in Computer Science ». Cham : Springer International Publishing. <https://doi.org/10.1007/978-3-319-05152-9>. Consulté le 4 février 2023.

Allen, Hunter L. 2016. « Threaded Applications with the roscpp API ». In Robot Operating System (ROS): The Complete Reference (Volume 1), sous la dir. de Koubaa, Anis, p. 51-69. Coll. « Studies in Computational Intelligence ». Cham : Springer International Publishing. <https://doi.org/10.1007/978-3-319-26054-9_3>. Consulté le 8 février 2023.

Bayer, Jan et Jan Faigl. 2019. « On Autonomous Spatial Exploration with Small Hexapod Walking Robot using Tracking Camera Intel RealSense T265 ». In 2019 European Conference on Mobile Robots (ECMR). (septembre 2019), p. 1-6. <https://doi.org/10.1109/ECMR.2019.8870968>.

Bor, Martin, John Edward Vidler et Utz Roedig. 2016. « LoRa for the Internet of Things ». p. 361-366. AUT : Junction Publishing. <https://eprints.lancs.ac.uk/id/eprint/77615/>. Consulté le 17 mars 2023.

Boren, Jonathan et Steve Cousins. 2011. « Exponential Growth of ROS [ROS Topics] ». IEEE Robotics Automation Magazine, vol. 18, no 1, p. 19-20. <https://doi.org/10.1109/MRA.2010.940147>.

Brooks, P. 2001. « Ethernet/IP-industrial protocol ». In ETFA 2001. 8th International Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No.01TH8597). (octobre 2001), p. 505-514 vol.2. <https://doi.org/10.1109/ETFA.2001.997725>.

Chen, Ching-Han, Ming-Yi Lin et Chung-Chi Liu. 2018. « Edge Computing Gateway of the Industrial Internet of Things Using Multiple Collaborative Microcontrollers ». IEEE Network, vol. 32, no 1, p. 24 32. <https://doi.org/10.1109/MNET.2018.1700146>.

Colgate, J. Edward, Witaya Wannasuphoprasit et Michael A. Peshkin. 1996. « Cobots: Proceedings of the 1996 ASME International Mechanical Engineering Congress and Exposition ». (1 décembre 1996), p. 433 439.

<http://www.scopus.com/inward/record.url?scp=0030402971&partnerID=8YFLogxK>. Consulté le 1 février 2023.

Cupek, Rafal, Adam Ziebinski et Marek Drewniak. 2017. « An OPC UA server as a gateway that shares CAN network data and engineering knowledge ». In 2017 IEEE International Conference on Industrial Technology (ICIT). (mars 2017), p. 1424 1429. <https://doi.org/10.1109/ICIT.2017.7915574>.

Deepthi, B., Venkata Ratnam Kolluru, George Varghese, Rajendraparasad Narne et Nerella Srimannarayana. 2020. « IoT based Smart Environment Using Node-Red and MQTT ». Journal of Advanced Research in Dynamical and Control Systems, vol. 12, p. 21-26. <https://doi.org/10.5373/JARDCS/V12I5/20201684>.

Eckhardt, Andreas, Sebastian Müller et Ludwig Leurs. 2018. « An Evaluation of the Applicability of OPC UA Publish Subscribe on Factory Automation use Cases ». In 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA). (septembre 2018), p. 1071-1074. <https://doi.org/10.1109/ETFA.2018.8502445>.

Efron, Bradley et Robert Tibshirani. 1985. « The Bootstrap Method for Assessing Statistical Accuracy ». Behaviormetrika, vol. 12, no 17, p. 1-35. <https://doi.org/10.2333/bhmk.12.17_1>.

Firdaus, Eko Nugroho et Alvin Sahroni. 2014. « ZigBee and wifi network interface on Wireless Sensor Networks ». In 2014 Makassar International Conference on Electrical Engineering and Informatics (MICEEI). (novembre 2014), p. 54-58. <https://doi.org/10.1109/MICEEI.2014.7067310>.

Freeman, Richard B. 2006. Is A Great Labor Shortage Coming? Replacement Demand in the Global Economy. <https://doi.org/10.3386/w12541>. Consulté le 6 février 2023.

Gemirter, Cavide Balkı, Çağatay Şenturca et Şebnem Baydere. 2021. « A Comparative Evaluation of AMQP, MQTT and HTTP Protocols Using Real-Time Public Smart City Data ». In 2021 6th International Conference on Computer Science and Engineering (UBMK). (septembre 2021), p. 542-547. <https://doi.org/10.1109/UBMK52708.2021.9559032>.

Greenwell, Bradley Boehmke & Brandon. [s d]. Chapter 2 Modeling Process | Hands-On Machine Learning with R. <https://bradleyboehmke.github.io/HOML/process.html>. Consulté le 22 mars 2023.

Gurram, Sunil Kumar et James M. Conrad. 2011. « Implementation of CAN bus in an autonomous all-terrain vehicle ». In 2011 Proceedings of IEEE Southeastcon. (mars 2011), p. 250 254. <https://doi.org/10.1109/SECON.2011.5752943>.

Hashim, Hairulazwan et Zainal Alam Haron. 2007. « A study on industrial communication networking: Ethernet based implementation ». In 2007 International Conference on Intelligent and Advanced Systems. (novembre 2007), p. 1111-1114. <https://doi.org/10.1109/ICIAS.2007.4658557>.

Hentout, Abdelfetah, Mustapha Aouache, Abderraouf Maoudj et Isma Akli. 2019. « Human–robot interaction in industrial collaborative robotics: a literature review of the decade 2008–2017 ». Advanced Robotics, vol. 33, no 15 16, p. 764 799. <https://doi.org/10.1080/01691864.2019.1636714>.

Huang, A S, E Olson et D C Moore. 2010. « LCM: Lightweight Communications and Marshalling ». In 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010). (Taipei, octobre 2010), p. 4057 4062. IEEE. <https://doi.org/10.1109/IROS.2010.5649358>. Consulté le 4 février 2023.

Katsikeas, Sotirios, Konstantinos Fysarakis, Andreas Miaoudakis, Amaury Van Bemten, Ioannis Askoxylakis, Ioannis Papaefstathiou et Anargyros Plemenos. 2017. « Lightweight & secure industrial IoT communications via the MQ telemetry transport protocol ». In 2017 IEEE Symposium on Computers and Communications (ISCC). (juillet 2017), p. 1193-1200. <https://doi.org/10.1109/ISCC.2017.8024687>.

Khanchuea, Kanitkorn et Rawat Siripokarpirom. 2019. « A Multi-Protocol IoT Gateway and WiFi/BLE Sensor Nodes for Smart Home and Building Automation: Design and Implementation ». In 2019 10th International Conference of Information and Communication Technology for Embedded Systems (IC-ICTES). (mars 2019), p. 1 6. <https://doi.org/10.1109/ICTEmSys.2019.8695968>.

Kinghorn, Dr Donald. 2020. « Does Enabling WSL2 Affect Performance of Windows 10 Applications ». In *Puget Systems*. <https://www.pugetsystems.com/labs/hpc/does-enabling-wsl2-affect-performance-of-windows-10-applications-1832/>. Consulté le 1 juin 2023.

Koubaa, Anis, éd. 2016. Robot Operating System (ROS). Coll. « Studies in Computational Intelligence ». Cham : Springer International Publishing. <https://doi.org/10.1007/978-3-319-26054-9>. Consulté le 22 avril 2021.

Lekić, Milica et Gordana Gardašević. 2018. « IoT sensor integration to Node-RED platform ». In 2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH). (mars 2018), p. 1-5. <https://doi.org/10.1109/INFOTEH.2018.8345544>.

Light, Roger A. 2017. « Mosquitto: server and client implementation of the MQTT protocol ». Journal of Open Source Software, vol. 2, no 13, p. 265. <https://doi.org/10.21105/joss.00265>.

Mishra, Biswajeeban et Attila Kertesz. 2020. « The Use of MQTT in M2M and IoT Systems: A Survey ». IEEE Access, vol. 8, p. 201071 201086. <https://doi.org/10.1109/ACCESS.2020.3035849>.

Noreen, Umber, Ahcène Bounceur et Laurent Clavier. 2017. « A study of LoRa low power and wide area network technology ». In 2017 International Conference on Advanced Technologies for Signal and Image Processing (ATSIP). (mai 2017), p. 1-6. <https://doi.org/10.1109/ATSIP.2017.8075570>.

Omron [s d]. « NX102 Machine Automation Controller | Omron ». <https://automation.omron.com/en/ca/products/family/index>. Consulté le 19 mars 2023.

Peng, Dao-gang, Hao Zhang, Li Yang et Hui Li. 2008. « Design and Realization of Modbus Protocol Based on Embedded Linux System ». In 2008 International Conference on Embedded Software and Systems Symposia. (juillet 2008), p. 275 280. <https://doi.org/10.1109/ICESS.Symposia.2008.32>.

Peralta, Goiuri, Markel Iglesias-Urkia, Marc Barcelo, Raul Gomez, Adrian Moran et Josu Bilbao. 2017. « Fog computing based efficient IoT scheme for the Industry 4.0 ». In 2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM). (mai 2017), p. 1 6. <https://doi.org/10.1109/ECMSM.2017.7945879>.

Quigley, Morgan, Brian Gerkey et William D. Smart. 2015. Programming Robots with ROS: A Practical Introduction to the Robot Operating System. O'Reilly Media, Inc., 86 p.

Quigley, Morgan, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler et Andrew Ng. 2009. « ROS: an open-source Robot Operating System ». In ICRA Workshop on Open Source Software. (1 janvier 2009).

Ramadhan, Aris, Muhammad Ary Murti et Leanna Vidya Yovita. 2013. « Implementation of networked control systems using programmable controller based Ethernet network ». In 2013 International Conference of Information and Communication Technology (ICoICT). (mars 2013), p. 22-27. <https://doi.org/10.1109/ICoICT.2013.6574543>.

Ramya, C. Muthu, M Shanmugaraj et R Prabakaran. 2011. « Study on ZigBee technology ». In 2011 3rd International Conference on Electronics Computer Technology. (avril 2011), p. 297 301. <https://doi.org/10.1109/ICECTECH.2011.5942102>.

Rojas, Maximiliano, Gabriel Hermosilla, Daniel Yunge et Gonzalo Farias. 2022. « An Easy to Use Deep Reinforcement Learning Library for AI Mobile Robots in Isaac Sim ». Applied Sciences, vol. 12, no 17, p. 8429. <https://doi.org/10.3390/app12178429>.

Sundaresan K., White G., Glennon S. [s d]. « Paper - Latency Measurement: What is Latency and How Do We Measure It? - NCTA Technical Papers ». <https://www.nctatechnicalpapers.com/Paper/2020/2020-latency-measurement>. Consulté le 19 février 2023.

Terrissa, Labib Sadek, Radhia Bouziane, Soheyb Ayad et Jean-François Brethé. 2016. « ROS-Based Approach for robot as a service in cloud computing ». (13 décembre 2016).

Tuza, Zoltán, János Rudan et Gábor Szederkényi. 2010. « Developing an integrated software environment for mobile robot navigation and control ». In 2010 International Conference on Indoor Positioning and Indoor Navigation. (septembre 2010), p. 1 6. <https://doi.org/10.1109/IPIN.2010.5647506>.

Tilkov, Stefan et Steve Vinoski. 2010. « Node.js: Using JavaScript to Build High-Performance Network Programs ». IEEE Internet Computing, vol. 14, no 6, p. 80-83. <https://doi.org/10.1109/MIC.2010.145>.

Wang, Huaimin, Gang Yin, Xiang Li et Xiao Li. 2015. « TRUSTIE: A Software Development Platform for Crowdsourcing ». In Crowdsourcing: Cloud-Based Software Development, sous la dir. de Li, Wei, Michael N. Huhns, Wei-Tek Tsai et Wenjun Wu, p. 165-190. Coll. « Progress in IS ». Berlin, Heidelberg : Springer. <https://doi.org/10.1007/978-3-662-47011-4_10>. Consulté le 22 mars 2023.

Watson, Venesa, Asmaa Tellabi, Jochen Sassmannahausen et Xinxin Lou. 2017. Interoperability and Security Challenges of Industry 4.0. Gesellschaft für Informatik, Bonn. <https://doi.org/10.18420/in2017_100>. Consulté le 15 mars 2023.

Yi, Yung et Sanjay Shakkottai. 2007. « Hop-by-Hop Congestion Control Over a Wireless Multi-Hop Network ». IEEE/ACM Transactions on Networking, vol. 15, no 1, p. 133-144. <https://doi.org/10.1109/TNET.2006.890121>.