POLITECNICO
MILANO 1863

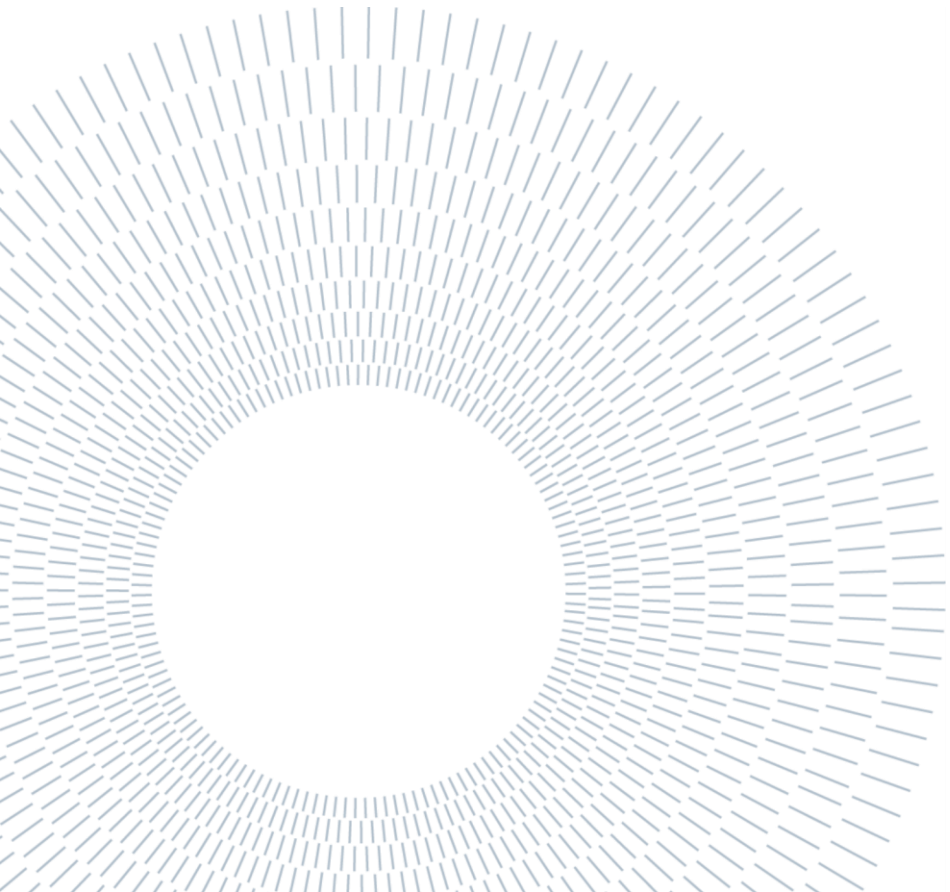
SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Investigation on Modbus protocol to develop an interoperable IIoT platform using Node-RED with MQTT gateway

TESI DI LAUREA MAGISTRALE IN
Telecommunication Engineering

Author: **Hiva Amiri**

Student ID: 10696153
Advisor: Prof. Alessandro Enrico Cesare Redondi
Co-advisors: Prof. Dieter Uckelmann
Hadi Adineh
Academic Year: 2021-22



Abstract

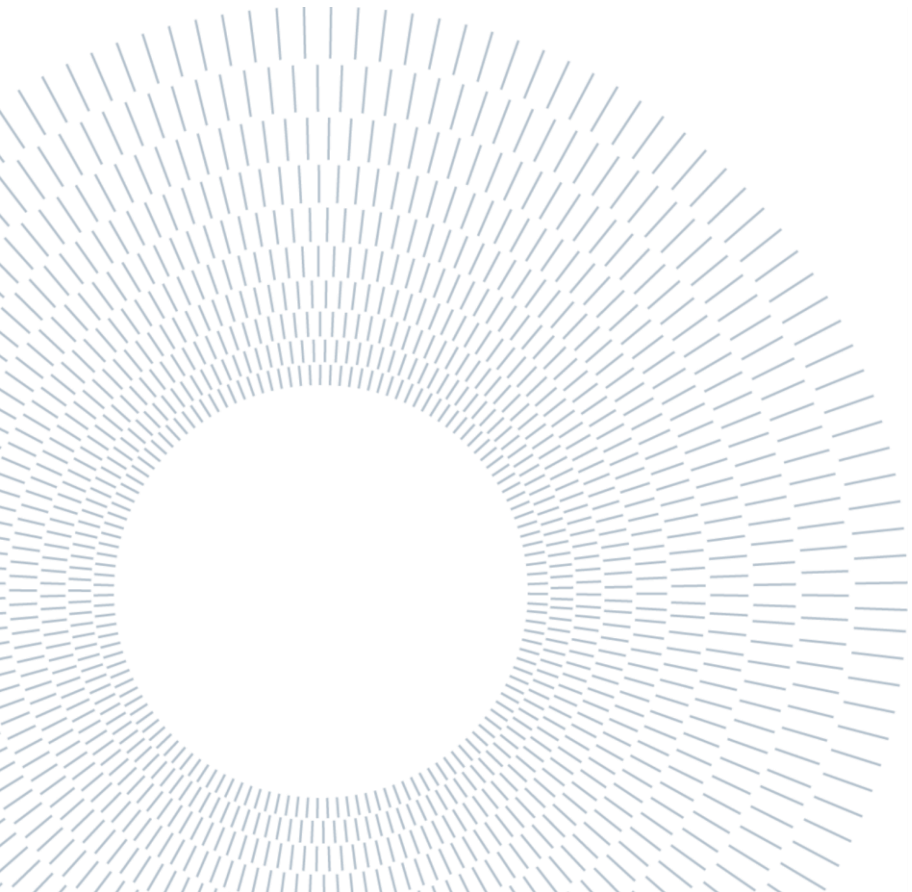
Various IoT protocols and IoT interfaces are being considered in order to develop a concept for a uniform exchange of IoT data. Interoperability of IoT systems, especially the exchange of structured data remains a challenge. In most scenarios, communication between systems is achieved by creating an extra layer of adapter or middleware solutions. For example, the IoT protocol MQTT is widely used in a variety of IoT applications and has a key role in the exchange of data between IoT systems. This is mainly due to the simplicity of the standard and its ease of use. The aim of the work is to develop an IIoT toolbox on the basis of a Raspberry Pi. This toolbox should enable a user to set up and connect individual IIoT systems. For this purpose, industrial communication protocols such as Modbus should be integrated to enable an exchange of data via an MQTT interface. In a further step, the safety of the entire system should be considered. The toolbox should allow connections to different devices or be able to simulate industrial devices (e.g. OPC-UA, Modbus), which can be used to create a reproducible environment to test a variety of common IoT protocols and IoT standards to exchange data generated by the toolbox over a MQTT interface.

Key-words: Modbus, MQTT, IIoT, IoT, protocol.

Abstract in lingua italiana

Sono allo studio diversi protocolli IoT e interfacce IoT al fine di sviluppare un concetto per uno scambio uniforme di dati IoT. L'interoperabilità dei sistemi IoT, in particolare lo scambio di dati strutturati, rimane una sfida. Nella maggior parte degli scenari, la comunicazione tra i sistemi si ottiene creando un ulteriore livello di adattatori o soluzioni middleware. Ad esempio, il protocollo IoT MQTT è ampiamente utilizzato in una varietà di applicazioni IoT e ha un ruolo chiave nello scambio di dati tra sistemi IoT. Ciò è dovuto principalmente alla semplicità dello standard e alla sua facilità d'uso. Lo scopo del lavoro è sviluppare un toolbox IIoT sulla base di un Raspberry Pi. Questa cassetta degli attrezzi dovrebbe consentire a un utente di configurare e connettere singoli sistemi IIoT. A tal fine, dovrebbero essere integrati protocolli di comunicazione industriale come Modbus per consentire uno scambio di dati tramite un'interfaccia MQTT. In una fase successiva, dovrebbe essere considerata la sicurezza dell'intero sistema. Il toolbox dovrebbe consentire connessioni a diversi dispositivi o essere in grado di simulare dispositivi industriali (ad es. OPC-UA, Modbus), che possono essere utilizzati per creare un ambiente riproducibile per testare una varietà di protocolli IoT comuni e standard IoT per scambiare dati generati dal toolbox su un'interfaccia MQTT.

Parole chiave: Modbus, MQTT, IIoT, IoT, protocol.



Contents

Abstract ..... **i**

Abstract in lingua italiana ..... **iii**

Contents ..... **v**

1. Introduction **1**

1.1 What is IoT, and how does IoT work? 1

1.2 What is IIoT, and how does IIoT work? 2

1.3 Differences between IIoT vs. IoT 2

2. Requirements review **5**

2.1 RevPi 6

2.1.1 RevPi Core 7

2.1.2 RevPi DIO 7

2.1.3 PiCtory 8

2.2 AUBO i10 robot 9

2.2.1 Cobot 9

2.3 Modbus 10

2.3.1 Addressing and messaging 12

2.3.2 Serial wiring and speed 13

2.3.3 Safety and cybersecurity 14

2.4 Node-RED 15

2.4.1 Key features of Node-RED: 16

3. Implementation **19**

3.1 Introduction 19

3.2 Technical information 19

3.2.1 Proper Infrastructure 20

3.3 System architecture 21

3.4 Detailed implementation 22
3.4.1 How is data stored in Standard Modbus? 25
3.4.2 What is a function code? 26
3.5 Integration 29
3.5.1 Inputs from TCP 32
3.5.2 Handling conflicts 32
4. Safety 35
4.1 Hazards 35
4.2 Hazard control 36
4.3 In case of this project 37
4.3.1 The AUBO i10 range of access 38
4.4 Safety Implementation 41
4.4.1 Safety handler of AUBO i10 robot 41
4.4.2 Safety handler of the integrated system 44
4.4.3 Software, unauthorized access 45
4.4.4 Secured integrated system 46
5. Experimental results 49
5.1 Stress test 50
5.1.1 Scenario 1 50
5.1.2 Scenario 2 52
5.1.3 Scenario 3 52
6. Conclusion and future development 55
Bibliography 57
A. Appendix A 59
A.1 Demonstration on Nodered-modbus-contrib nodes 59
List of Figures 63
List of Tables 65
Acknowledgements 66

1. Introduction

The Internet of Things, or IoT, and the Industrial Internet of Things, or IIoT, are two of the revolution's key technologies. You've almost certainly heard these terms before and may even be familiar with some IoT and IIoT technology. IoT and IIoT enable businesses to run more efficiently, make more informed decisions, and generate new revenue streams by connecting various devices and pieces of equipment via the Internet.

1.1 What is IoT, and how does IoT work?

The Internet of Things (IoT) is a network of connected devices that are capable of communicating with one another and providing data to users via the Internet. IoT devices can connect to the Internet and frequently include sensors that collect data. While an IoT device can be useful on its own, when combined with others, it becomes even more valuable.

As more types of equipment become Internet-connected, the IoT expands in size. The Internet of Things encompasses a diverse array of devices. It can encompass anything from factory machinery to electrical substations to buildings and infrastructure, with the number of connected devices increasing daily. The Internet of Things is used by manufacturers, energy companies, city governments, and a variety of other types of organizations.

The Internet of Things enables you to collect data automatically from a variety of functions, such as how much energy a building's lighting consumes or how much water flows through a wastewater treatment plant. Through the Internet, IoT solutions and devices can transmit the data they collect to a central system. Managers can then use this data to help them make more informed decisions. By employing data analysis techniques, you can delve deeper into the data in order to uncover new insights and forecast future outcomes.

Additionally, you can use IoT technology to automate certain pieces of equipment and processes within your business. Intelligent sensors enable equipment to automatically adjust its operation to optimize energy consumption, traffic flow, and

more. When they detect a particular input, smart sensors take action or signal for an action to occur. Motion-activated lighting is a simple example. When those sensors detect movement, they activate the lights. Additionally, smart sensors can detect abnormal conditions or device operation and notify operators of potential issues.

1.2 What is IIoT, and how does IIoT work?

IIoT is a subcategory of the Internet of Things. The term refers to Internet of Things (IoT) technology that is used in industrial settings, specifically manufacturing facilities. IIoT is a critical component of Industry 4.0, the fourth industrial revolution. Industry 4.0 places a premium on smart technology, data, automation, interconnectivity, and other technologies and capabilities. These technologies are fundamentally altering how factories and industrial organizations operate.

IIoT has a number of the same applications and benefits as IoT. Smart sensors can be integrated into manufacturing equipment, energy systems, and infrastructure such as piping and wiring. Through the data they collect and the advanced functionality they enable, these sensors can assist industrial businesses in increasing their efficiency, productivity, and employee safety, among other metrics.

The IIoT enables improved machine-to-machine communication and provides plant managers with data that provides a more complete picture of their facility's operation. Industrial companies can keep a closer eye on their energy, water, and other resource consumption, as well as when and how much they produce, by collecting granular data on a continuous basis. Operators can then make manual adjustments to optimize their operation, or the equipment can do so automatically.

Businesses can save significant amounts of energy, water, and resources while maintaining or increasing productivity levels through this continuous optimization. When used in this manner, IIoT can assist businesses in achieving their lean manufacturing goals. Additionally, you can use IIoT to inform predictive maintenance programs, accelerate product development, and accomplish other business objectives. [1]

1.3 Differences between IIoT vs. IoT

When defining IoT and industrial IoT, it's critical to understand that IIoT is a subset of IoT technology that refers specifically to industrial activities. While it shares many

of the same concepts and functions as traditional IoT applications, there are some distinctions between IIoT and traditional IoT applications. Review some of the following differences between the IIoT and IoT below:

- **Market focus:** The Internet of Things spans a broad range of industries and users. IoT technology may be used by consumers as well as professionals in industries such as healthcare, business, and the public sector. Due to the fact that IoT is used in so many different sectors and by so many different users, it tends to focus on more general applications. In comparison, IIoT technology is only used in industrial settings by professionals in related fields, implying a narrower market focus. The IIoT has a number of critical applications in power plants, oil and gas refineries, and manufacturing facilities.
- **Goals:** Users of IIoT and IoT technology frequently have slightly different objectives. Typically, IoT deployments aim to boost efficiency, improve health and safety, and create better experiences. Typically, IIoT is more concerned with the first two goals and is less user-centric. Consumers do not use IIoT in their daily lives; rather, it is a component of industrial processes.
- **End devices:** Due to the fact that IIoT and IoT have distinct focuses and objectives, they frequently make use of distinct devices. IIoT devices are intended to provide users with data about their machinery, and are intended to work in conjunction with existing machinery rather than independently. Controllers and Programmable Logic Controllers are examples of these devices. IoT devices are typically everyday items that can be used independently, such as smart thermostats, smartwatches, and personal assistants. Occasionally, you'll come across IoT smart sensors and other devices that are used to improve infrastructure.
- **Risk of failure:** When IoT devices and technology fail, the risk is relatively low due to the small scale at which these devices are used. Generally, IoT devices are not used for critical processes that could result in death or serious injury if they fail. In comparison to IoT, IIoT devices and technology failures can be more dangerous, as IIoT technology is network-connected and can result in life-threatening situations when heavy machinery fails.
- **Development needs:** When businesses develop new IoT devices, they typically aim to improve the convenience of a user's daily life. The development effort is primarily focused on increasing comfort, as consumer demand for ease of use is high. Typically, IIoT development is focused on creating new devices that improve the efficiency of clients' operations. Due to the fact that industrial plants must optimize their processes to remain

competitive, IIoT developers leverage data metrics to create devices that assist businesses in cutting costs and increasing efficiency.

- **Compatibility with legacy systems:** By and large, IoT devices are not required to be backward compatible with legacy systems. Due to the fact that these devices frequently operate independently, device designers are not required to make them backward compatible. In comparison, IIoT devices typically require compatibility with a variety of legacy devices and machines found in industrial plants. Due to the fact that these plants frequently use equipment without digital interfaces or functionality, many IIoT devices must assist legacy equipment in providing digital data and accepting IT system commands.
- **Environmental requirements:** Typically, IoT devices must operate in everyday environments, with their designs built to withstand normal temperatures and other environmental pressures. Due to the harsh environments in which IIoT devices operate, such as energy plants, factories, and oil refineries, they must be more durable and reliable. As a result, manufacturers of IIoT devices typically design their products to withstand moisture, radio interference, and extreme temperatures in order to ensure consistent results. [1]

2. Requirements review

Standardization is a significant barrier to digitized manufacturing. Standards are critical for ensuring data exchange between machines, systems, and software throughout a networked value chain as a product progresses through an interconnected, intelligent factory. Additionally, they would be critical for ensuring that robots can be easily integrated into the manufacturing process via plug-and-play. However, if data and communication protocols are proprietary or only recognized nationally, only a subset of companies' equipment will be compatible; competition and trade will suffer, and costs will rise. On the other hand, independent, internationally recognized standards for communication protocols, data formats, and interfaces can ensure interoperability across sectors and countries. Thus, we can promote widespread adoption of Industry 4.0 technologies and ensure open markets for European manufacturers and products on a global scale. This thesis emphasized the importance of anticipating standard requirements and expediting their development. In this regard, it is hoped to conduct research on various industrial communication protocols in order to develop an integrated platform capable of unifying all fieldbus protocols. [2]

In an automated system, various industrial communication protocols are used because each one serves a distinct business and technical purpose. An OEM may select a particular protocol because their vendor offers a high-functioning off-the-shelf solution that is customized to their specifications. On the other hand, an end user may choose to work with a particular protocol due to its geographic availability or its focus on the requirements of their industry. There are a variety of reasons why an independent software developer or an original equipment manufacturer selects specific communication protocols. [3]

The various types of industrial communication protocols are

- Ethernet/IP
- BACnet
- Modbus RTU
- Modbus TCP
- DeviceNet
- OPC UA

- ProfiNet
- ProfiBus
- IO-Link
- EtherCAT
- CC-Link
- CAN Open
- IEC 61131-3
- ASI-Interface
- LonWorks

Protocols can be public or private. Any entity or business can use open protocols. They are adaptable and guard against vendor lock-in. On the other hand, proprietary protocols are designed specifically for a particular network or system. They include customized features and upgrades. The disadvantage is that you may be required to pay licensing fees and work exclusively with protocol-compliant devices.

Protocols for communication can be wired or wireless. USB, SPI, 12C, and Ethernet are just a few of the physical media that the wired protocol can communicate over. Wi-Fi, LoRa, LTE, and Bluetooth Low Energy are just a few of the wireless protocol mediums available.

To imply a diverse range of devices with varying resource constraints, communication protocols, and data structures, it is critical to connect these disparate devices to one another and to the Internet in order to construct systems composed of these devices.

The Web of Things is a concept that enables interoperability between IoT and IIoT devices and platforms by utilizing Thing Descriptions to describe devices and their capabilities. Once a device understands the capabilities of another, it can automatically interact with it and form systems of devices, regardless of the communication protocol, platform, or other constraints. [4]

The purpose of this thesis is to contribute to the development of an integrated platform that enables devices with disparate communication protocols to communicate with one another, as well as to investigate automatic system compositions using a web-based monitoring system powered by could technologies.

2.1 RevPi

Revolution Pi, or RevPi for short, is an open, modular, and low-cost industrial PC based on the popular Raspberry Pi computers. It is housed in a slim DIN-rail chassis and can be expanded seamlessly with a variety of suitable I/O modules and fieldbus

gateways. The 24V powered modules connect in seconds via an overhead connector and can be configured easily using a graphical configuration tool. All of the options available on Raspberry Pis are also available here, albeit in a more convenient manner.

2.1.1 RevPi Core

RevPi core is powered by the Raspberry Pi Compute Module, it is the core part of every Revolution Pi modular system.

Figure 2.1-1: Revpi Core module

2.1.2 RevPi DIO

Multiple digital I/O modules can be connected to the base module RevPi Core to transform Revolution Pi into an industrial control unit. Three versions of the I/O modules are available. Each has the same 28-pin I/O connector on the front (connector with two rows of 14 pins each – two 14-pin socket connectors with spring clamp contacts suitable for connecting up to 1.5 mm2 stranded hookup wires are included). Apart from the standard configuration with 14 digital inputs and 14 digital outputs, there are two special configurations with 16 digital inputs or 16 digital outputs.

Figure 2.1-2: Revpi IO modules

2.1.3 PiCtory

PiCtory is a web-based application that allows you to communicate between your RevPi and connected devices using a configuration file. This configuration file is created by means of PiCtory. [5]

The main functions of a configuration file are:

- Communicating type and position of the expansion modules to the base module
- Communicating mutually the basic settings of your RevPi and connected devices
- Using the configuration values of your RevPi in other applications

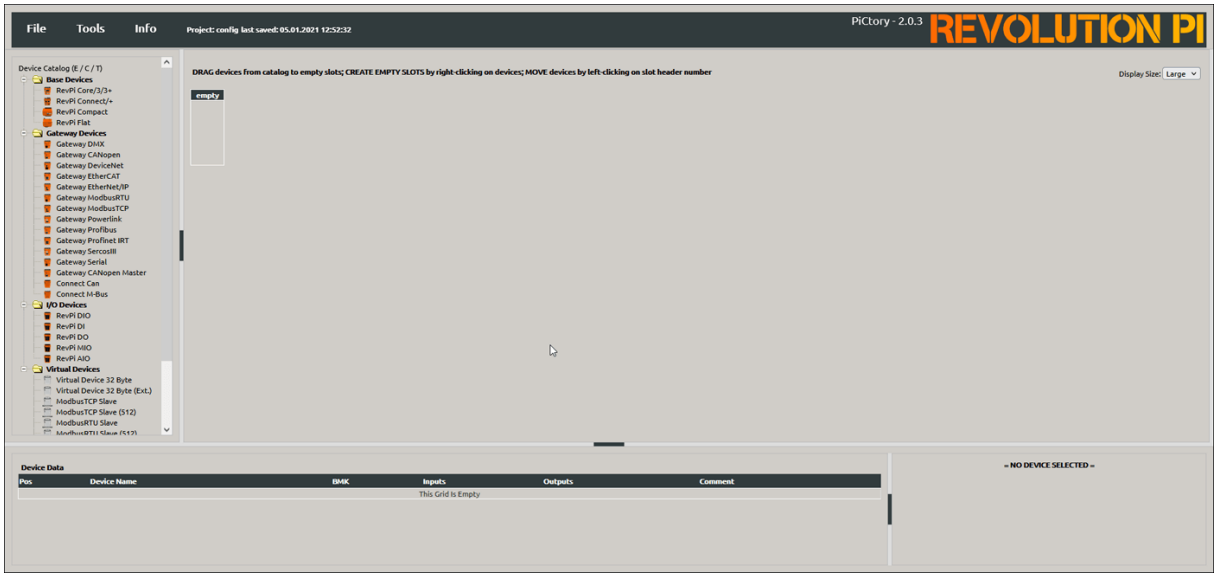


Figure 2.1-3: PiCtory user interface

2.2 AUBO i10 robot

The AUBO i10 is a collaborative hand guide-to-teach robot that enables rapid programming and adaptation to industrial process requirements. It is defined by a high degree of safety when working without a fence in a human-operated environment. Supports open architecture and is easily integrated into existing production systems. A vision system can be integrated into the six-axis robot with a payload capacity of up to 10kg in the 1350mm range. This robot is typically used in corporate laboratories or academic research on robots. [6]

2.2.1 Cobot

Cobots, or collaborative robots, are a new breed of manufacturing robots that are designed to work alongside humans rather than in their own space. They're just as effective — if not more so — than their larger counterparts, with the added benefit of sharing workspaces with humans. This property is advantageous, particularly in factories with limited floor space, as it eliminates the need for dedicated bubbles to keep everyone safe.

Because cobots are not necessarily autonomous, human operators are still required, and certain safety precautions must be taken.

Rather than being instructed by a programmer, cobots are frequently taught by example. An operator physically directs the bot's movements, directing it to complete its assigned tasks. The cobot can then recall which tasks it needs to complete and repeat them flawlessly. The video below illustrates how to train a cobot. [7]

Figure 2.2-1: AUBOi10 robot

2.3 Modbus

Modbus is a communication protocol that is widely used in process and factory automation for transmitting data between electronic devices over serial lines (original version) or Ethernet. While "Modbus" is an open protocol, it is a registered trademark of Schneider Electric USA, Inc. (current owner of the Modicon brand). Schneider Electric was a founding member of the Modbus.org organization, which was formed to advance the use of Modbus. This article provides an overview of Modbus and its fundamental functions—Modbus.org has extensive coverage of Modbus, including the Modbus specifications for various types of Modbus, software, testing, and interface code. Additionally, the Internet provides tutorials and detailed information on specific Modbus implementations for individual devices. [8]

The Modbus serial protocol (in its original form) is a master/slave protocol, with a master controlling Modbus data transactions and multiple slaves responding to the master's requests to read from or write to the slaves. Modbus TCP, alternatively

referred to as Modbus TCP/IP, employs a client/server architecture. These network architectures are shown Figure 2.3-1 and Figure 2.3-2.

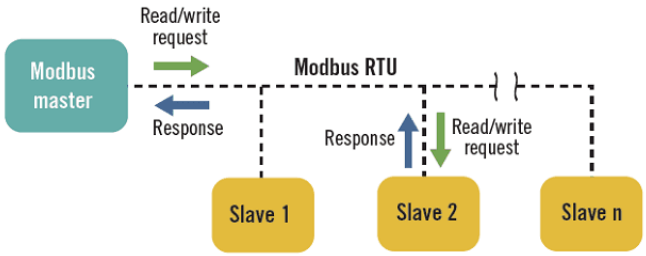


Figure 2.3-1: Modbus RTU

There is one master and up to 247 slaves in a standard Modbus serial network, each with its own unique slave address. Modbus TCP is typically implemented over an Ethernet network, and data transactions initiated by a Modbus client are routed via an IP address to a Modbus server.

Modbus is available in a variety of configurations, including Serial RTU, Serial ASCII, TCP/IP, and UDP/IP. Modbus dialects such as Enron, Daniel, and Pemex Modbus have developed as a result of people modifying standard Modbus to support floating-point data, long integer data, and other data requirements. Reading the Modbus interface and slave documentation is critical for understanding and implementing these types of Modbus networks, as well as for safely mixing devices from different manufacturers in the same Modbus network.

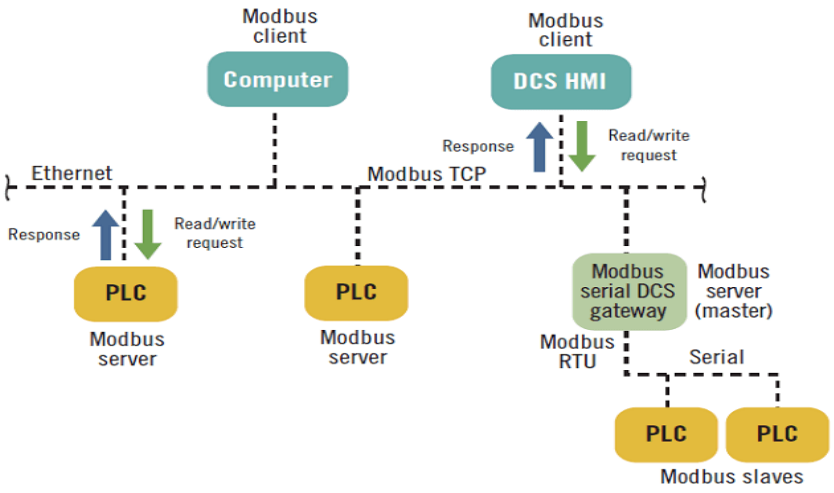


Figure 2.3-2: Modbus TCP/IP

Modbus is an application-layer protocol that is not dependent on the medium used to transmit data. The master/client initiates data transactions by requesting data from or writing data to the slave/server. The master/client controls the data transactions, and there is no data-by-exception transmission in standard Modbus. The data is stored in 16-bit registers, which can contain either discrete on/off values or 16-bit integer values. To represent floating data or long integer values, some implementations use two or more integer registers. A Modbus serial master can request diagnostic data from the slave, and the slave/server can send error codes to the master/client if they detect an error with the request they received. Modbus data transactions contain only a function code, register addresses, and data; it is up to the master/client and slave/server to interpret the data.

2.3.1 Addressing and messaging

Modbus memory addressing is typically organized around 16-bit registers that contain sixteen coils or on/off (0/1) states or integer values stored in sixteen 16-bit registers (input/output or holding registers). While some devices employ their own Modbus addressing scheme, standard Modbus addressing is illustrated in section 0.

Modbus messaging is composed of two components known as an application data unit (ADU) and a protocol data unit (PDU). Modbus messages contain the slave/server address of the slave/server is involved, a function code, data start addresses, and the data being sent to (written) or read back from (read) the master/client, as well as an error checksum (CRC/LRC/Checksum).

The serial Modbus PDU is limited in size by the 256-byte size constraint inherited from the first Modbus serial network implementation. Modbus slave addresses are limited to values between 1 and 255. The user has access to addresses 1-247, while addresses 248-255 are reserved.

A typical Modbus serial data transaction is shown in Figure 2.3-3. The Modbus TCP data transactions are identical except for the server address, an IP address, some Ethernet overhead, and a different error checksum. Modbus data can include starting addresses, data quantities or counts, and the actual data being read or written. If the Modbus slave/server encounters an error while processing the master/client request, the slave/server will return an error to the master/client.

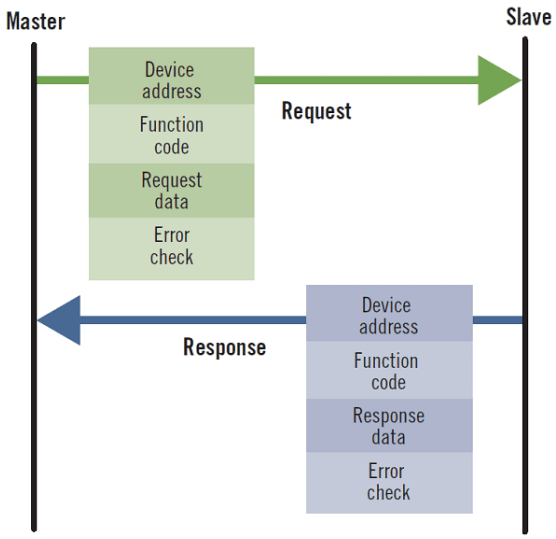


Figure 2.3-3: Modbus data transaction

The Modbus PDU is typically wrapped into the Ethernet package in the Modbus TCP/IP message format and consists of the Modbus function code and the Modbus data request. Typically, the slave address and error code (CRC) are not required because the Modbus TCP/IP packet is routed by the network to the desired IP address (unless a connection to a serial network is required), and the error check is performed as part of the Ethernet packet.

Modbus data transactions are defined by function codes, which indicate to the Modbus slave/server the type of data transaction that is occurring. Public codes, user codes, and reserved codes are all types of function codes. The Modbus.org community has defined well-defined public function codes that are guaranteed to be unique and validated. It is possible to implement user function codes, but they are not supported by the Modbus specification. There is no guarantee that the code for the user function will be unique. Reserved function codes are those that are currently being used by some companies for legacy products and are therefore not available to the public use.

2.3.2 Serial wiring and speed

Modbus serial has traditionally collected data from Modbus slaves via RS-232 and RS-485 wiring. Modbus was originally based on RS-232, but the requirement to connect multiple slaves quickly expanded Modbus to include multidrop wiring methods. Although RS-422 was used, RS-485 is more capable and is more widely used. A Modbus master can communicate with up to 247 Modbus slaves (limited by

the capability of the RS-422/485 driver), which can be PLCs, smart devices, DCS, or remote telemetry units (RTUs), which is how Modbus RTU got its name.

To connect more than two devices over a distance of more than 50 feet, a network of multidrop RS-485 or RS-422 devices should be used. RS-485 is by far the most popular protocol for multidrop networks, as it supports up to 32 nodes without repeaters over a range of up to 4,000 feet (1,200 m).

The baud rate (or bits per second) at which Modbus messages are transmitted is called the baud rate, and all devices on the network must use the same baud rate, typically 9,600-19,200 Baud, but network speeds of up to 115 kb/s are not uncommon. In general, the faster the transmission speed, the shorter the cable should be and the more critical it is to have the proper network end termination resistors to minimize reflections (equal to resistance component of the cable characteristic impedance).

2.3.3 Safety and cybersecurity

Any incorrect data in control systems such as a DCS can potentially result in a safety incident, regardless of whether a safety instrumented system (SIS) is involved. As a result, we must be concerned with the data integrity of Modbus data transactions, ensuring that the correct data is transmitted and received correctly. Modbus standard provides error detection via parity, CRC/LRC, and checksums, as well as diagnostics via the slave/server or master/client interfaces. These safeguards should be sufficient for routine data transactions, but not against cyberattacks or internal security breaches.

Modbus serial has a lower cybersecurity risk due to its hardwired nature than Ethernet-based client/server communications using Modbus TCP, particularly when the Ethernet is connected to a larger process control network that can be connected to the corporate enterprise network and/or the Internet. Modbus serial can be vulnerable to cybersecurity attacks on the master's control system or when connected to a Modbus TCP network. Using Ethernet-based Modbus TCP without additional security can significantly increase the likelihood of a security breach.

The Modbus TCP network's cybersecurity exposure can be further reduced by utilizing security appliances or firewalls with deep packet inspection that are specifically designed for Modbus. Between a Modbus TCP network and a Modbus serial network, as well as between any remote access to a Modbus Ethernet network, a security appliance/Modbus firewall should always be installed. (The term "Modbus firewall" refers to a firewall or security appliance designed specifically for Modbus data transactions and equipped with deep packet inspection, whitelisting, and other Modbus-specific security features.) Figure 2.3-4 illustrates a simple network diagram

for Modbus TCP/Serial with SIS systems, segmented into zones and conduits in accordance with IEC 62443. There are numerous possible network configurations, and this example illustrates only the Modbus portion of the zones and conduits. [9]

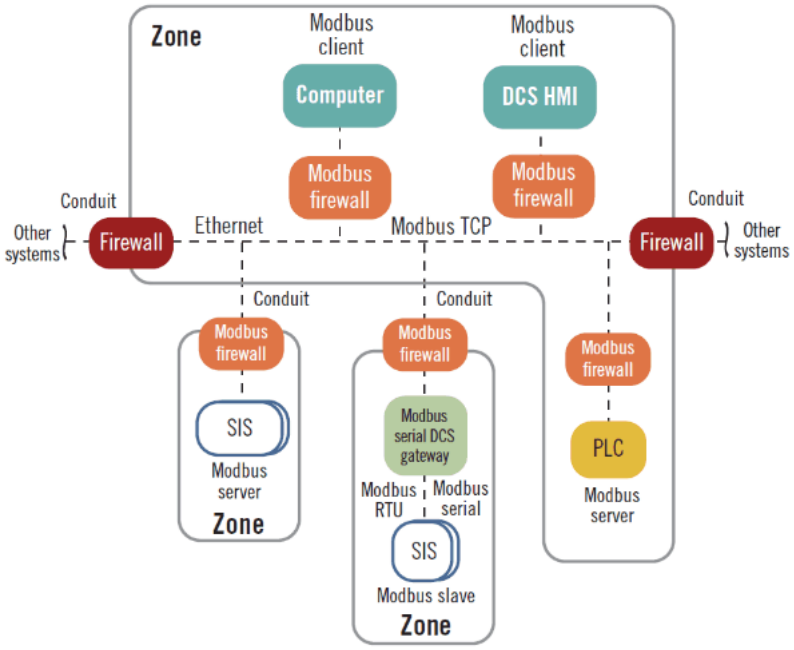


Figure 2.3-4: This simple network example for Modbus TCP/serial with SIS systems is broken down into zones and conduits per IEC 62443. There are many different possible configurations, and only the Modbus portion of the zones and conduits are shown.

2.4 Node-RED

Node-RED is an open-source programming tool for creatively and easily connecting hardware devices, APIs, and online services. It is primarily a visual tool for Internet of Things applications, but it can also be used for other applications to rapidly assemble flows of various services.

It is constructed using nodes, which are blocks of JavaScript-based software code. These prebuilt nodes mitigate risk associated with software development and accelerate time to market. The nodes are visually dragged and dropped to simplify, automate, and scale IIoT application development.

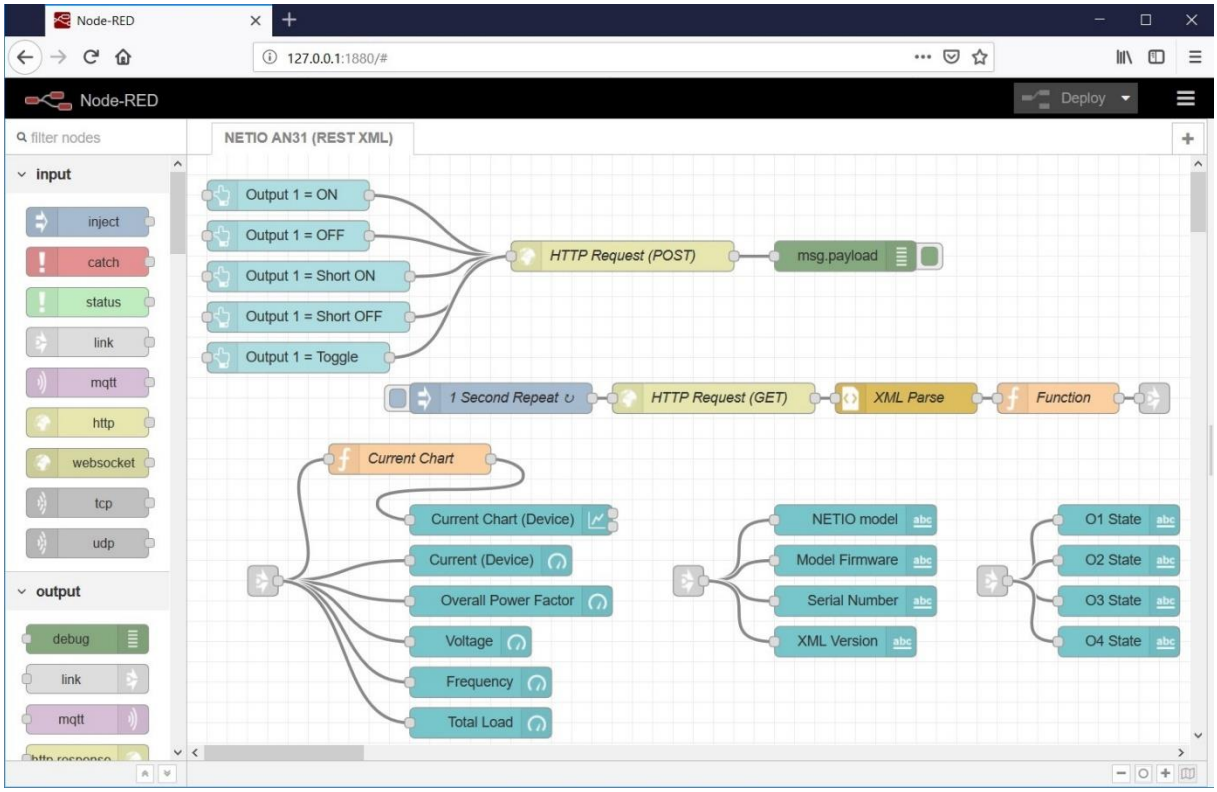


Figure 2.4-1: Node-RED environment

Node-RED is a cross-platform application that runs on Windows, Mac, and Linux computers, as well as industrial gateways and I/O. Certain Advantech offerings, such as the EKI-1241NR, include Node-RED pre-installed, and the company is soon to release a series of Node-RED-compatible IoT gateways. Additionally, certain Moxa US-series RISC-based devices support Node-RED, as demonstrated in our white paper 'How to Install Node-RED on a Moxa device'.

The ability to deploy this type of logic at the network's edge enables data to be processed, reformatted, or sorted prior to being transmitted further up the network or to the cloud. This reduces the amount of traffic sent — a concept dubbed Fog Computing.

2.4.1 Key features of Node-RED:

- Supports browser-based flow editing making it user friendly, accessible and visual
- It is built on Node.js, which is a none-blocking, lightweight I/O model, making it lightweight and efficient

- Flows created in Node-RED are stored using JSON, and can imported and exported and shared with ease
- Node-RED can be run locally
- Ability to run in cloud environments like Bluemix, MS-Azure, FRED etc
- Node library is continuously growing
- Built-in support for MQTT, Modbus, CoAP, AMQP, BLE, OPC UA, and even mDNS protocols
- Simple user interface creation

3. Implementation

3.1 Introduction

This chapter discusses the system's implementation. Section 3.2 contains technical information about the system, such as the system's architecture, software design, and decision-making processes. Section 3.3 describes the system's structure, illustrating the various directories and file organization. It goes over the directory structure of the lesson directories and describes the platform's organization and design. Section 3.4 summarizes the completed courseware. It describes the various components of the protocol translation, such as the Modbus to MQTT conversion, monitoring, and command transmission. It discusses the safety concerns that will be thoroughly explained in the following chapter.

3.2 Technical information

Industrial communication is a relatively developed field. This project's software and development process could be the subject of an entire dissertation. This section attempts to summarize the development process in a concise manner, emphasizing the most critical points.

The Modbus protocol was chosen to construct the project since the main purpose of this thesis is to provide a platform to unify industrial communication technologies. On the other hand, MQTT was a solid choice as the primary communication protocol for the unification. Modbus is one of the many industrial communication protocols encountered in the field. In the Introduction chapter, there is a brief description of it, while we will go into detail about it in this chapter.

Because MQTT is utilized in most sections of the platform, such as monitoring, transmitting commands, and the communication tunnel between different system hardware resources, MQTT, as the key player in the game, has the duty of unifying the entire system.

A variety of functions are required to read, transform, and send data to the proper output based on requirements. In recent years, the component-based platform with off-the-shelf and open-source functionalities has matured to the point where it is no longer necessary to construct everything from scratch.

Node-RED, a pioneering platform for implementing the notion, would be an excellent choice for moving forward with the project. This platform is built on

Node.js and includes many pre-defined functionalities called "nodes". The nodes can be changed according to the requirements of a particular task.

What makes the concept more challenging is to make the entire system real-time. In the industrial fields, real-time communication is crucial, and different field devices should be aware of each other's stats in milliseconds. The subject becomes more important when safety comes into play. The safety procedure should be fast enough to avoid any danger to the on-site operators in close contact with field devices like arm robots. The safety procedure is separated from the monitoring and command sending channels to achieve this goal.

On the one hand, the integrated system performs reading, processing and sending commands to the field devices. On the other hand, the safety system is doing its job in parallel to gain a close to zero delay in case of danger. Meanwhile, the data from the safety system is also sent to the integrated system to become aware of the danger. To have a better demonstration, consider an arm robot as an example. The robot is under the direct supervision of the integrated system to do a specific task. The safety system has noticed the danger, and the stop signal is sent to the arm robot directly. The same signal is sent to the integrated system to perform further decisions when the danger goes away.

3.2.1 Proper Infrastructure

Implementing the integrated system requires choosing the proper hardware that satisfies the project's needs. Based on the nature of the system, the underlying hardware should support a variety of inputs and outputs since the system needs to receive and send data over it. The cost-efficiency should be taken into account as well.

RevPi is a company developing a series of hardware based on RaspberyPi mini computers. The RevPi Core 3 was chosen to proceed with the project because it has a wide range of choices to satisfy the project needs. This hardware plays a central role in the integrated system by receiving, processing and sending commands.

The RaspberryPi 4 is another part of the system's architecture that handles the monitoring with a user-friendly interface to the operators. It can be connected to several RevPi core3 to monitor and provide user interfaces.

The physical infrastructure between RevPis and the Raspberry pi is Ethernet, using the legacy network equipment. From the software point of view, the MQTT plays the main role in relaying messages among devices. It is reliable, fast, secure and easy to implement.

3.3 System architecture

The designed template contains many different components. This section provides a succinct overview of the structure of the system. Figure 3.3-1 shows the overall system architecture.

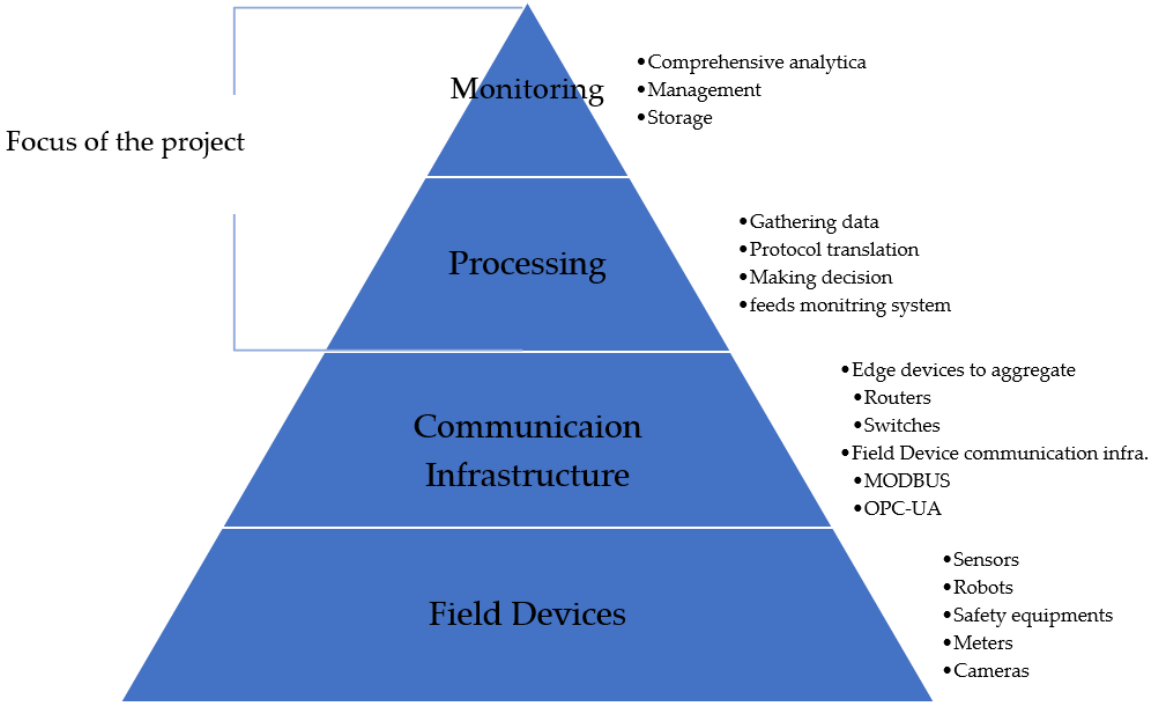


Figure 3.3-1: The overall system architecture

At the lowest level, the system contains a wide range of devices that can generate data and receive a command from the controller. Meters, sensors, cameras, and safety equipment are data generators mainly, and the system controls robots, actuators, drives, etc., by making decisions based on them. The communication protocols might differ in any field device, which can be a problem. The main goal of the integrated system can show its importance at this point. This system helps to integrate all the protocols used in the field. The first two layers of the architecture shown in Figure 3.3-1 are not under the focus of this project, while the translation of protocols and relaying data are done in higher levels of the architecture.

To go more in detail in the focus of this project, the architecture of the integrated system is shown in the Figure 3.3-2.

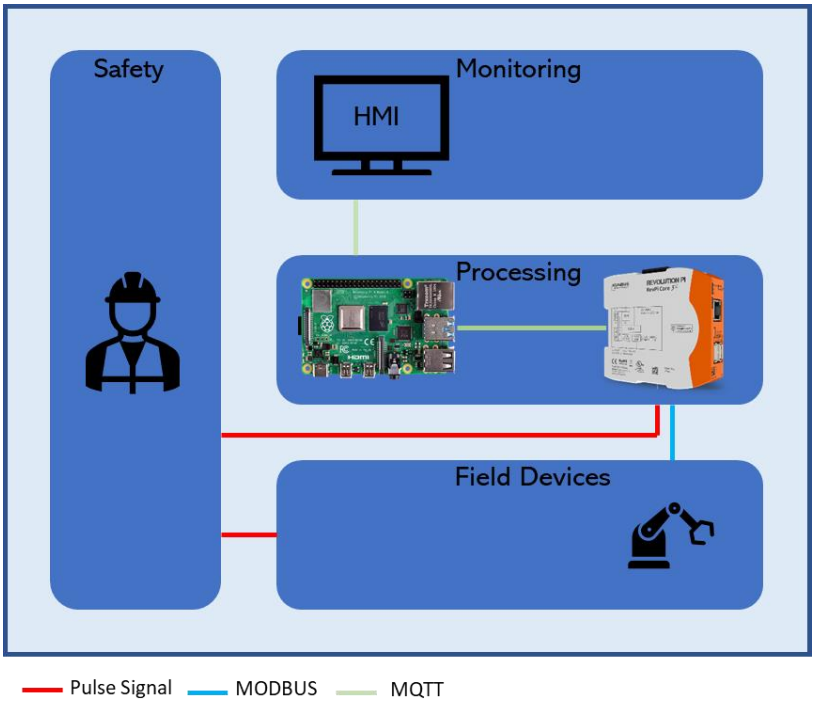


Figure 3.3-2: More detail in the architecture of the project

It is tried to design the system in such a way that the safety system works in parallel with the integrated system. Current architecture assures that nothing interrupts the safety system. Designing the safety system and the different scenarios that perfectly fit the project will be discussed in chapter 4. This chapter's focus is mainly on the processing level of the system.

TCP and RTU are two forms of Modbus protocols, which were described in detail in Chapter 2. Unlike the former, which makes use of the well-known TCP connections across an Ethernet infrastructure, the latter is implemented on a Serial port using the RS-232 and RS-485 interfaces. The Modbus TCP protocol will be examined in this project, and the programs will be developed in accordance with the protocol type indicated above.

3.4 Detailed implementation

The Modbus protocol is a server/client-based protocol, which means that the server-side of the system can be implemented either in the field devices or in the integrated system, depending on the application. It has been agreed to include the server as part of the integrated system. The reason for this is that a manufacturing line has a number of field devices, all of which are continually generating and transmitting

data. The server would be put under extra pressure if messages were relayed. Consequently, this would result in less efficiency when carried out by field equipment.

On the other hand, each field device has its own architecture for storing data in its registers, which is unique to that device. Due to the fact that this project is intended to be integrated, the field devices should adhere to a system-specific design. Based on these two considerations, the RevPi Core3 would be the most suitable platform to develop the Modbus server.

All other field devices that connect with one another using the Modbus protocol are considered to be on the client-side of the Modbus protocol. They establish a connection to the server based on configurations that have been predefined.

The AUBO-i10 robot, as an example of a field device, is capable of communicating through Modbus TCP as well as RTU. Figure 3.4-1 depicts the Modbus TCP menu on the robot's teach pendant when the Modbus TCP is selected. For starters, connecting the robot as a Modbus client to the integrated system as a Modbus server and reading and writing values into its registers would be necessary.

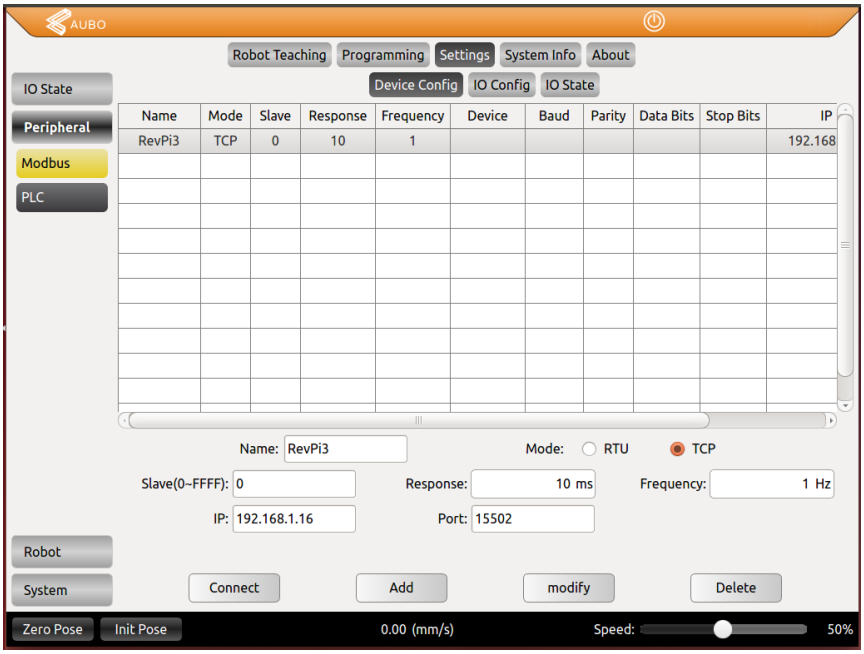


Figure 3.4-1: Modbus configuration menu on AUBO i10

The RevPi Core3 has Node-RED installed by default. After activating the Node-RED on the RevPi Core3 web interface, the Node-RED can be reached by typing the IP address of the RevPi Core3 and the port number 1881 on the browser.

`http://RevPiCore3IP:1881`

By default, Node-RED does not have any function node to support MODBUS. The Modbus functions can be installed by opening the Manage Palette menu, searching for "node-red-contrib-modbus" and installing it. After installation, the nodes regarding the Modbus functions appear on the window's left side. The node "modbus-server" can be used to implement the modbus server on RevPi Core3. In the Figure 3.4-2, the property of the "modbus-server" node is shown.

The hostname 0.0.0.0 means that all incoming connections are accepted. The Modbus server only listens to port number 10500, while the order of the registers can be seen in the rest.

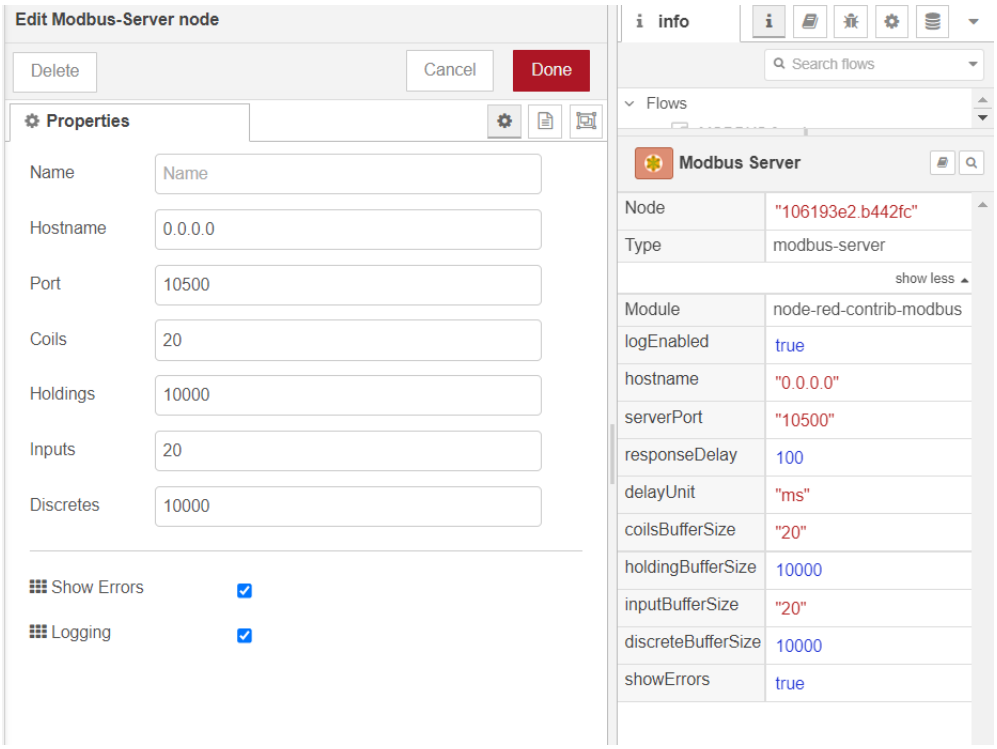


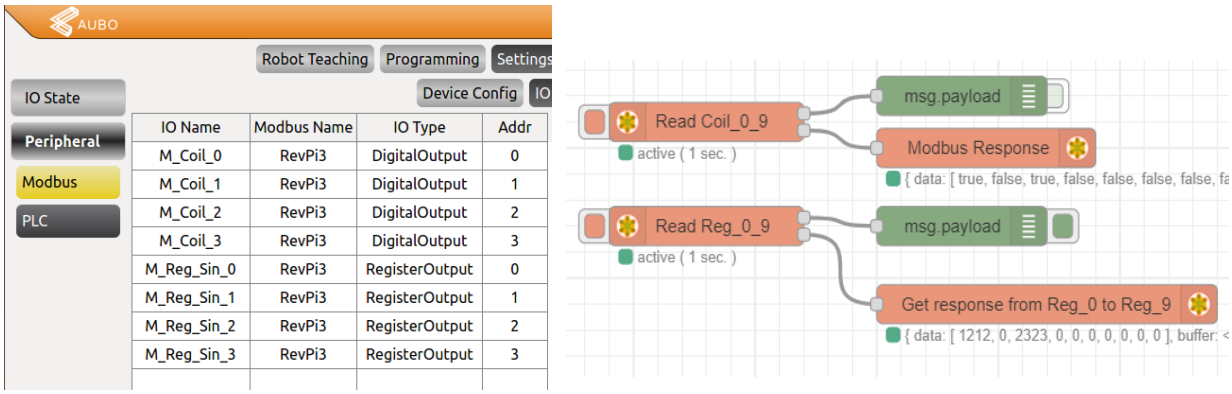
Figure 3.4-2: Configuration page of "modbus-server" node on Node-RED

Each client is limited to communicating with a single port. If there are several clients in the field, more than one server is required in order to open as many ports as are required by the clients. Each server node has its own memory scope, which means they do not share the same memory address scope as the other server nodes. If the field devices require a standard memory scope, a function should be made to handle the requirement for the shared memory space. The use of global variables can be used to implement a shared memory space.

Once the Modbus server implemented and configured, the same configuration should be applied to the clients. The Figure 3.4-3(a) shows the configured Modbus

client on AUBO i10. Now the AUBO i10 is connected to the integrated system using Modbus protocol over a TCP connection.

To take a simple implementation as an example, Figure 3.4-3(b) shows a modbus-read node on RevPi Core3 and Figure 3.4-3(a) is the AUBO i10 client which is connected to the server. The "Send_RevPi Coil_0" sends the generated command by inject node. In this case the commands are 0 and 1. The "Read_RevPi_Coil" node reads the coil in a determined time interval.



(a) AUBO i10 as a client

(b) RevPi Core3 as a server

Figure 3.4-3: Modbus server and client

The Modbus does not have a specific memory partitioning. Based on needs, each field device has its specific memory partitioning. In this project, the memory can be partitioned as it is needed. In order to allocate memory addresses, the concept of different memories is elaborated in the following.

3.4.1 How is data stored in Standard Modbus?

A total of four different tables store information in the Slave device. Two tables store discrete values (coils) that can be turned on and off, and two tables store numerical values (registers). Both read-only and read-write tables are present in each of the coils and registers. [10]

- Each table has 9999 values.
- Each coil or contact is 1 bit and assigned a data address between 0000 and 270E.
- Each register is 1 word = 16 bits = 2 bytes and also has data address between 0000 and 270E.

Table 3-1: Standard addresses in Modbus

Coil/Register Numbers	Data Addresses	Type	Table Name
1-9999	0000 to 270E	Read-Write	Discrete Output Coils
10001-19999	0000 to 270E	Read-Only	Discrete Input Contacts
30001-39999	0000 to 270E	Read-Only	Analog Input Registers
40001-49999	0000 to 270E	Read-Write	Analog Output Holding Registers

Due to the fact that they do not appear in the actual messages, coil/register numbers can be thought of as location names. Messages contain Data Addresses, which are used to send information. For example, the first Holding Register, number 40001, has the Data Address 0000. The difference between these two values is the offset. Each table has a different offset. 1, 10001, 30001 and 40001.

3.4.2 What is a function code?

The Function code is contained in the second byte sent by the Master. This number instructs the slave which table to access as well as whether to read from or write to the table in question.

Table 3-2: Function codes of Modbus protocol

Function Code	Action	Table Name
01 (01 hex)	Read	Discrete Output Coils
05 (05 hex)	Write single	Discrete Output Coil
15 (0F hex)	Write multiple	Discrete Output Coils
02 (02 hex)	Read	Discrete Input Contacts
04 (04 hex)	Read	Analog Input Registers
03 (03 hex)	Read	Analog Output Holding Registers
06 (06 hex)	Write single	Analog Output Holding Register
16 (10 hex)	Write multiple	Analog Output Holding Registers

Read Input Status (FC=02)

Request

This command is requesting the ON/OFF status of discrete inputs # 10197 to 10218 from the slave device with address 17.

```
11 02 00C4 0016 BAA9
```

11	The Slave Address (11 hex = address17)
02	The Function Code 2 (read Input Status)
00C4	The Data Address of the first input to read. (00C4 hex = 196 , + 10001 offset = input #10197)
0016	The total number of coils requested. (16 hex = 22, inputs 197 to 218)
BAA9	The CRC (cyclic redundancy check) for error checking.

Response

11 02 03 ACDB35 2018

11	The Slave Address (11 hex = address17)
02	The Function Code 2 (read Input Status)
03	The number of data bytes to follow (22 Inputs / 8 bits per byte = 3 bytes)
AC	Discrete Inputs 10204 -10197 (1010 1100)
DB	Discrete Inputs 10212 – 10205 (1101 1011)
35	2 space holders & Discrete Inputs 10218 – 10213 (0011 0101)
2018	The CRC (cyclic redundancy check).

The more significant bits contain the higher Discrete inputs. This shows that input 10197 is off (0) and 10204 is on (1). Due to the number of inputs requested, the last data field 35 contains the status of only 6 inputs. The two most significant bits in this data field are filled in with zeroes.

3.4.2.1 Modbus RTU over TCP

For the uninitiated, this is a Modbus RTU message that has been wrapped in a TCP/IP packet and transmitted over a network rather than serial lines. As a result, the Server does not have a SlaveID because it instead makes use of an IP Address. See more details in chapter 2.3.2.

3.4.2.2 Modbus TCP

A Modbus Messaging Implementation Guide provided by Schneider Automation outlines a modified protocol specifically for use over TCP/IP. See more details in capter 2.3.2.

3.4.2.3 ADU & PDU

Aside from the main differences between serial and network connections stated above, there are a few differences in the message content.

Starting with the Modbus RTU message and removing the SlaveID from the beginning and the CRC from the end results in the PDU, Protocol Data Unit.

Here is an example of a Modbus RTU request for the content of analog output holding registers # 40108 to 40110 from the slave device with address 17.

11 03 006B 0003 7687

11	The SlaveID Address (17 = 11 hex)
03	The Function Code (read Analog Output Holding Registers)
006B	The Data Address of the first register requested. (40108-40001 = 107 =6B hex)
0003	The total number of registers requested. (read 3 registers 40108 to 40110)
7687	The CRC (cyclic redundancy check) for error checking.

Removing the SlaveID and CRC gives the PDU:

03 006B 0003

3.4.2.4 MBAP Header

A new 7-byte header called the MBAP header (Modbus Application Header) is added to the start of the message. This header has the following data:

- Transaction Identifier: 2 bytes set by the Client to uniquely identify each request. These bytes are echoed by the Server since its responses may not be received in the same order as the requests.
- Protocol Identifier: 2 bytes set by the Client, always = 00 00

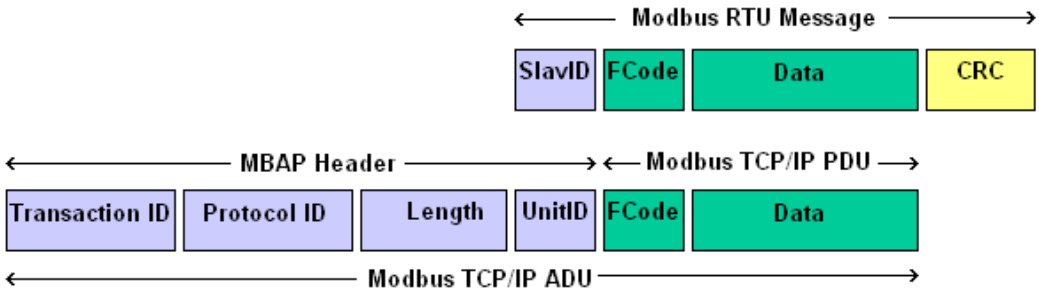


Figure 3.4-4: Header of Modbus RTU and Modbus TCP

- Length: 2 bytes identifying the number of bytes in the message to follow.
- Unit Identifier: 1 byte set by the Client and echoed by the Server for identification of a remote slave connected on a serial line or on other buses.

Summary

The equivalent request to this Modbus RTU example

11 03 006B 0003 7687

0001	Transaction Identifier
0000	Protocol Identifier
0006	Message Length (6 bytes to follow)
11	The SlaveID Address (17 = 11 hex)
03	The Function Code (read Analog Output Holding Registers)
006B	The Data Address of the first register requested. (40108-40001 = 107 =6B hex)
0003	The total number of registers requested. (read 3 registers 40108 to 40110)
7687	The CRC (cyclic redundancy check) for error checking.

In the preceding section, a summary of the Modbus protocol was presented, which will be useful in dealing with the issues and obstacles that arise while sending and receiving messages via the protocol. In the next part, we will discuss how to integrate and link protocols. [10]

3.5 Integration

What has been elaborated in the previous section should be used to translate Modbus messages to MQTT. As mentioned before, the Modbus protocol does not have a specific memory address. Each device with a Modbus interface has its memory registers addressed differently from any other device. Different types of addresses as shown in Table 3-1 with the standard memory addressing. The integrated platform as a unique software is flexible to address registers of Modbus. As an end-user, the addresses can be configured based on desire. This project has stuck to the standard memory allocation since the RevPi Core3 does not have any memory restriction.

The integration among protocols would be straightforward by taking advantage of the Node-RED platform. Various libraries are available to deal with Modbus protocol, where any library could be installed on Node-RED as a palette. The palette

named "node-red-contrib-modbus" was used to handle Modbus communication. For instance, the node "modbus-write" is used to write in the registers. It supports four-function codes related to writing in the registers.

- FC 05: Force single coil
- FC 06: Preset single register
- FC 15: Force multiple coils
- FC 16: Preset multiple registers

The address of the register should be indicated in the next step. This address is the

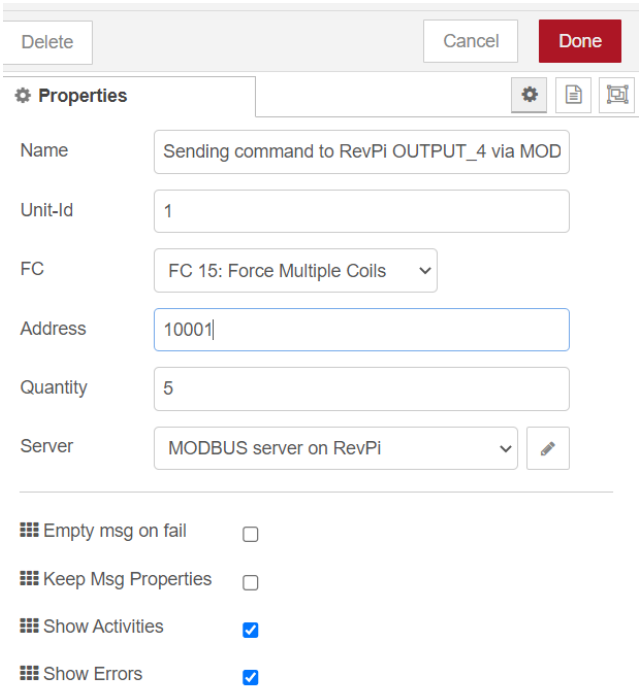


Figure 3.5-1: Properties of “modbus-write” node

first coil or register in the row. If the multiple coils or registers is chosen, then the number of register or coils that are needed to write should be mentioned as well. All these parameters are available while the node is configured. The Figure 3.5-1 shows the configuration of the "modbus-writer" node.

As long as the "modbus-writer" node receives data, it executes the code for every single input. The input should be in a format that is understandable for Modbus registers. To make an example, the coils only understand true or false values, while the holding register can save a word equal to two bytes.

Function nodes can do the translation of data received from other sources. A function node has the ability to execute a written code once an input is received. In code 1, a

part of the code in the function node is provided. The code receives the data from MQTT as a string. The string is converted to an understandable value that Modbus can handle, and it is sent to the first output of the function node. The output of the function node is connected to a "modbus-writer" node that was mentioned above.

Algorithm 1 Understandable values to Modbus

```
if (global.get("ESPLC1") == 1 || global.get("ESPLC2") ==1 || global.get("ESUS")
== 1){
   global.set("ESGlobal", 1);
   global.set("EM", 1);
   msg.payload = false;

   return [msg,[],[]];
}
```

The translation of data is done by saving the data in the memory of the RevPi Core3 and then sending it to the destination via Modbus. In algorithm 1, data of 0 is written to the register of the field device.

In cases where we need more write and read nodes to transfer data, the "modbus-writer" and "modbus-reader" would not be suitable. These two nodes are suitable for a maximum of 10 nodes on one communication configuration. The "modbus-flex-write" and "modbus-flex-read" are flexible to handle more than ten nodes on one communication configuration. The input of these nodes is a line of code containing the data that the system will send to the field devices. A function node is used to write the code and send it to the output.

Algorithm 2 Function node code example for single write

```
msg.payload = { value: msg.payload,
'fc': 5,
'unitid': 1,
'address': 0 ,
'quantity': 1
}
return msg
```

The flex nodes are easier to deal with reading and writing data because the function code that is used beforehand can be configured depending on the codes implemented.

The "node-red-contrib-modbus" module contains 12 different nodes in general to support all the functions regarding Modbus communication. The whole demonstration on function nodes is written in appendix A.

3.5.1 Inputs from TCP

Now, setting up a Modbus server and exchanging data among the server and clients is clear. AUBO i10 robot is an example of a wide range of field devices that exploit Modbus for communication. Each field device might have a different way to configure the Modbus client, but the configuration concept would be the same as the examples elaborated in the section.

The following section tries to implement TCP protocol to put a step further in integration. The Node-RED can also handle TCP connections either as a server or a client. A TCP server using the built-in "tcp-in" node was implemented to handle clients who prefer to operate the system through TCP. For instance, operator "op1" wants to ask the robot to perform the task "welding" she connects to the integrated system using the HMI application installed on the tablet. At the same time, the only available connection is TCP. In this scenario, the connection between the operator and the integrated system is not MQTT, but it is TCP.

3.5.2 Handling conflicts

The commands might be sent from any inputs, but the final result would be a Modbus command to the field devices. The question that might come to mind is that when we have deferent inputs, the different commands from different sources would be a conflict if sent to the field devices. This problem was also solved by taking advantage of the semaphore concept inherited from operating system programming.

3.5.2.1 Semaphores

It is possible to solve the critical section problem using two atomic operations, wait and signal, which are used for process synchronization. Semaphores are integer variables that are used for this purpose The following are the definitions of wait and signal.

- Wait

If its argument S is positive, the wait operation decrements its value. No action is taken if S is negative or zero.

- Signal

The signal operation increments the value of its argument S.

3.5.2.2 Types of Semaphores

Counting semaphores and binary semaphores are the two most common types of semaphores. These are the specifics, as follows:

- Counting Semaphores

These are semaphores for integer values with an unbounded value domain. These semaphores are used to coordinate resource access, with the number of semaphores equal to the available resources. When resources are added, the count of semaphores is automatically increased; when resources are removed, the count is decremented.

- Binary Semaphores

Binary semaphores are similar to counting semaphores except that their value ranges between 0 and 1. The wait operation is successful only when the semaphore is set to 1, whereas the signal operation is successful when the semaphore is set to 0. Occasionally, binary semaphores are easier to implement than counting semaphores. [11]

The concept of binary semaphores was implemented to avoid any conflict in performed instructions. The HMIs are also synced in real-time. For example, when the operator "op1" sends a command to robo1 to perform a task over the TCP based HMI, at the same time, the operator "op2" can see that the robot1 is performing a particular task on her MQTT based HMI.

4. Safety

When robots are used in the workplace, workplace robotics safety is a component of occupational safety and health. This category includes both traditional industrial robots and emerging technologies such as drone aircraft and robotic exoskeletons that can be worn. Collisions, crushing, and injuries caused by mechanical parts are all examples of accidents. Physical barriers, safe work practices, and proper maintenance are all examples of hazard controls. Numerous robots in the workplace are industrial robots used in manufacturing. According to the International Federation of Robotics, between 2017 and 2020, factories are expected to add 1.7 million new robots. [12], [13] Collaborative robots, personal care robots, construction robots, exoskeletons, autonomous vehicles, and drone aircraft are all emerging robot technologies (also known as unmanned aerial vehicles or UAVs). [16]

Automation advancements (e.g., fixed robots, collaborative and mobile robots, and exoskeletons) have the potential to improve working conditions in manufacturing, but also to introduce workplace hazards. [17] 56% of robot injuries are classified as pinch injuries, while 44% are classified as impact injuries. According to a 1987 study, line workers are the most vulnerable, followed by maintenance workers and programmers. The majority of injuries were caused by poor workplace design and human error. [14], [15]

4.1 Hazards

Numerous hazards and injuries can arise when robots are used in the workplace. Certain robots, most notably those used in traditional industrial settings, are extremely fast and powerful. This increases the risk of injury, as a single swing from a robotic arm, for example, can result in serious bodily injury. [18] When a robot malfunctions or requires maintenance, additional risks arise. A worker working on the robot may sustain an injury due to the unpredictable nature of a malfunctioning robot. For instance, a jammed motor may occur on a robotic arm that is part of a car assembly line. A worker attempting to fix the jam may be struck by the arm as it becomes unjammed. Additionally, if a worker is standing in an overlap zone with nearby robotic arms, he or she may sustain an injury from other moving equipment. [15]

Robotic accidents can be classified into four categories: impact or collision accidents, crushing and trapping accidents, mechanical part accidents, and other accidents. Accidents involving impact or collision typically occur as a result of malfunctions

and unanticipated changes. Accidents involving crushing and trapping occur when a worker's body part becomes trapped or caught on robotic equipment. When a robot malfunctions and begins to "break down," mechanical part accidents can occur, resulting in the ejection of parts or exposed wire, which can result in serious injury. Other accidents are simply accidents that occur as a result of working with robots. [18]

There are seven sources of hazards that are associated with human interaction with robots and machines:

- human errors
- control errors
- unauthorized access
- mechanical failures
- environmental sources
- power systems
- improper installation

Human errors can range from a single incorrect line of code to a loose bolt on a robotic arm. Numerous hazards can be attributed to human error. Control errors are inherent and, in most cases, are neither controllable nor predictable. Unauthorized access hazards occur when an unfamiliar person enters a robot's domain. Mechanical breakdowns can occur at any time, and the behavior of a faulty unit is typically unpredictable. Environmental sources are anything in the environment that can cause a robot to malfunction, such as electromagnetic or radio interference. Pneumatic, hydraulic, or electrical power systems all have the potential to malfunction, resulting in fires, leaks, or electrical shocks. Improper installation is self-evident; a loose bolt or exposed wire can result in inherent dangers. [18]

4.2 Hazard control

There are several methods for preventing injuries through the use of hazard controls. Risk assessments can be conducted at each stage of a robot's development. Risk assessments can assist in determining the status of a robot, its level of maintenance, and whether repairs are imminent. By being aware of a robot's status, injuries and hazards can be avoided. [8] Preventative measures can be taken to minimize the risk of injury. These can include physical barriers, guard rails, and presence-sensing safeguarding devices, among others. The majority of the time, awareness devices are used in conjunction with safeguarding devices. Typically, they consist of a network of rope or chain barriers equipped with lights, signs, whistles, and horns. Their purpose is to warn workers or other personnel of potential dangers. [18]

Additionally, operator safeguards can be implemented. Typically, these incorporate safeguarding devices to protect the operator and minimize the risk of injury. Additionally, when an operator is in close proximity to a robot, the robot's operating speed can be reduced to maintain complete control over the situation. This is accomplished by switching the robot to manual or teach mode. Additionally, it is critical to inform the robot's programmer about the type of work the robot will perform, how it will interact with other robots, and how it will interact with an operator. [18] It is also critical to maintain robotic equipment properly in order to minimize hazards. Maintaining a robot ensures that it continues to operate properly, minimizing the risk of a malfunction.

4.3 In case of this project

The purpose of this project is to integrate safety procedures to prevent unauthorized access to field devices. Unauthorized access can take the form of physical or remote access. In the event of physical unauthorized access, a variety of sensors and applications may be beneficial in establishing a safe environment on the job.

Due to the fact that arm robots are attached to a platform and have a limited range of access referred to as robot application access, the following safety zone can be defined.

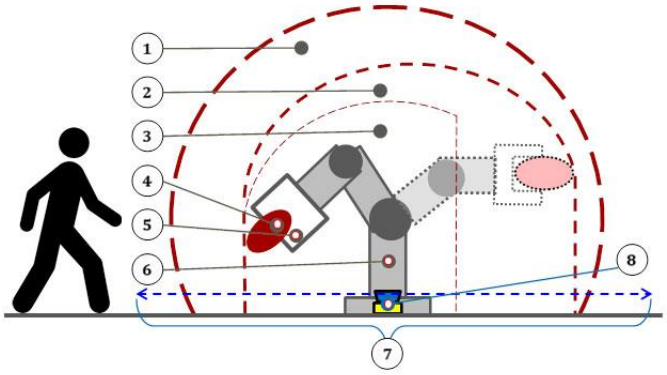


Figure 4.3-1: Safety zone of an arm robot

1. maximum space
2. restricted space
3. operating space
4. workpiece
5. end-effector
6. manipulator

7. safeguarded space
8. protective device or barrier(safety scanner shown)

4.3.1 The AUBO i10 range of access

The workspace of the manipulator, as shown in Figure 4.3-2, is a sphere of radius 1350mm except the cylindrical space directly above and directly below the robot base. When choosing the installation position, be sure to consider the cylindrical space directly above and directly below the robot base must avoid moving the tool into this cylindrical space as much as possible. In practical application, the range of rotation of joint 1 to joint 6 is -175°~ +175°. [6]

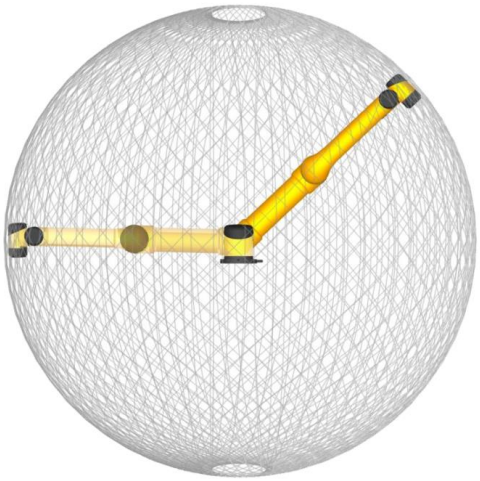


Figure 4.3-2: The workspace of the manipulator

To restrict access to the robots' range of motion, a fence door was installed to prevent unauthorized access and to prevent any injuries that may occur while the robots are operating. The robot's working environment is depicted in Figure 4.3-3.

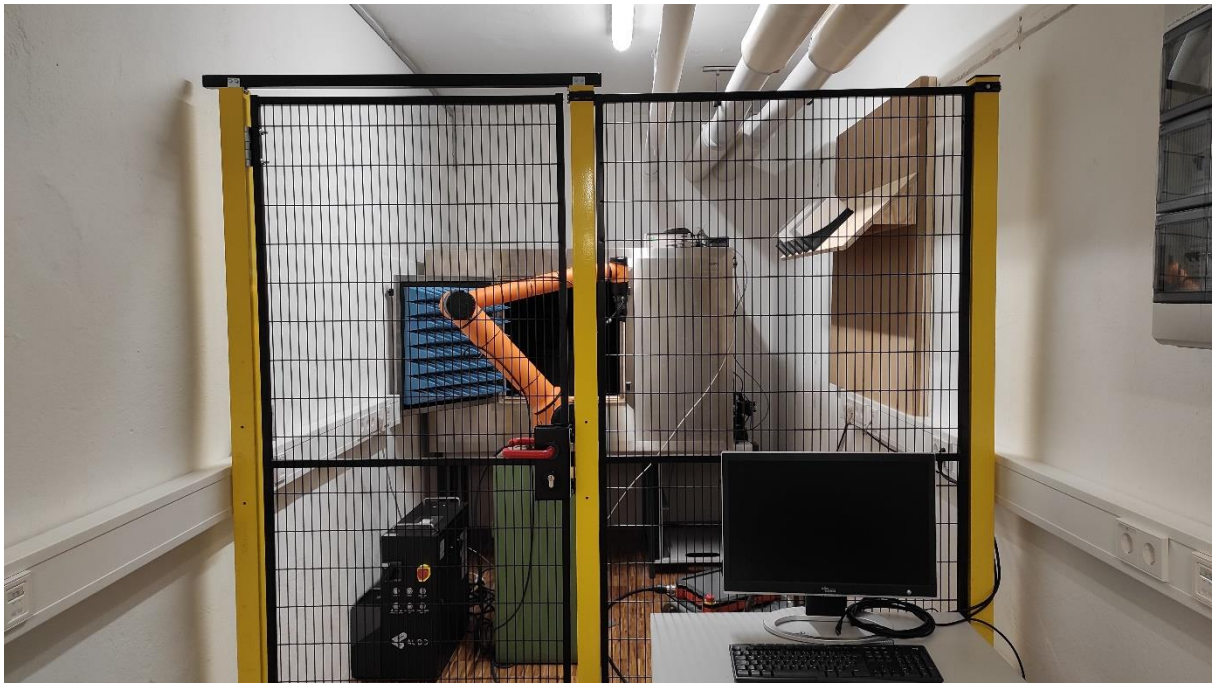


Figure 4.3-3: Working environment of AUBO i10

The safety scenario of the robot has been developed in such a way that a set of sensors are frequently measuring the environmental variables. These sensors are

- Door sensor
  - In case if the fence door is opened.
- Laser scanner
  - To measure any obstacle in the zone of danger
- Temperature and humidity sensor
  - To avoid any over heating of equipments
- Image processing
  - To recognize any danger in the environment

Each output of the above-mentioned devices should be managed in a proper way. The AUBO i10 has a set of functions in case of emergency. The Table 4-1 shows the I/O interface of the aubo that can be exploited to handle emergencies.

Table 4-1: Emergency I/O interface of AUBO i10 robot

Input	SI00	SI10	External Emergency Stop	SI04	SI14	Enabling Device
	SI01	SI11	Safeguard Stop	SI05	SI15	Operational Mode
	SI02	SI12	Reduced Mode Input	SI06	SI16	Hand Guiding Enable
	SI03	SI13	Safeguard Stop Reset	SI07	SI17	System Stop Input
Output	SO00	SO10	Robot Emergenccy Stop	SO04	SO14	Not Reduced Mode
	SO01	SO11	Robot Moving	SO05	SO15	System Error
	SO02	SO12	Robot Not Stopping	SO06	SO16	BACKUP (Unavailable for User)
	SO03	SO13	Reduced Mode	SO07	SO17	BACKUP (Unavailable for User)

The safety procedure is defined as follow

- Door sensor trigger:

Because entering the environment does not pose a threat, the door sensor trigger is connected to SI02, which causes the robot's speed to be reduced. In this case, the robot's movement speed is reduced to a pre-defined speed in a portion of its normal speed, rather than the entire normal speed.

- Laser scanner

Different zones can be handled by the laser scanner. This means that the operator has the ability to define several zones of danger based on the distance between the zones. For example, in this project, the laser scanner has two danger zones that it must navigate through. When the obstacle is within 2 meters of the robot, the SI01 is triggered, causing the robot to enter safeguard mode. The robot's movement is temporarily halted as a result of this strategy for dealing with the emergency. Once the obstacle has been removed, the robots can resume their operations.

When an obstacle is approached from a distance of 1.3 meters or more, the SI00 will be activated, on the other hand. The SI00 is an emergency stop, which means that the robot will stop working and will be turned off as soon as the input is triggered. In this case, a robot operator should be sent to the site in order to handle the situation.

- Image processing

When image processing techniques are used to deal with emergencies and maintain safety, a variety of options can be provided. A live video stream from a camera

installed in the laboratory can be used to identify obstacles in close proximity to the robot, as well as any unauthorized access to the laboratory.

It is expected that there would be more than one source that triggers the emergency stop input while the robot has only one input for external emergency stop. In this case, all the emergency stops sources should be connected in a series of closed-circuit. Any trigger from any source causes an emergency stop signal to the robot. The robot in this approach is blind when it faces an emergency. The reason is that the robot does not know which one of the sources has triggered the emergency signal. All the emergency signals are connected to the robot and the integrated system simultaneously. However, the connection between safety devices and the integrated system is not in series but parallel. The source of the emergency stop signal can be identified since the safety devices are connected separately to the integrated system.

4.4 Safety Implementation

This section tries to implement a scenario to maintain safety and avoid any hazards that can happen to the AUBO i10 robot. There is 7 type of hazards in general, but in this project, we try to go into detail to handle unauthorized hazards.

This kind of hazard can be found in two forms of physical or software access to the equipment. Two sets of equipment are used to control and monitor physical access to the robot. The door sensors are an excellent tool to recognize if the fence door is open or not. It was mentioned in the previous section that when the door sensor triggers the system, the signal is transferred to both the robot and the integrated system simultaneously.

4.4.1 Safety handler of AUBO i10 robot

The implementation of safety handler is different based on each specific field device. In this section we go in detail to see how it is possible to handle any type of safety signals that triggers the robot.

The schematic circuit to handle the alerts from the door sensor is shown in the Figure 4.4-1. The 16XT1 and 16XT2 are two inputs for the reduced mode. The user can use these interfaces to control the manipulator entering the reduced mode. In this mode, the motion parameters (joint speed, TCP speed) of the manipulator are limited to the user-defined reduced mode range.

In parallel, the laser scanner scans the environment to detect any obstacle close to the robot's working space. It was mentioned above that safety laser scanners have the ability to be programmed to separate different zones. The zones are distances that we

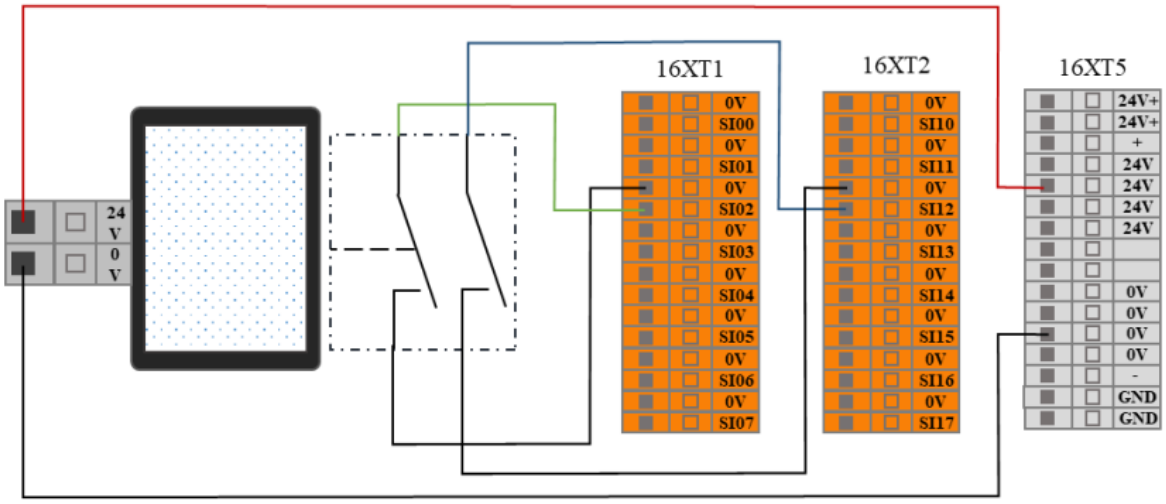


Figure 4.4-1: Reduced mode input connection triggered by door sensor

define. In the Figure 4.4-2 you can see an example of a safety zone defined by a random safety laser scanner.

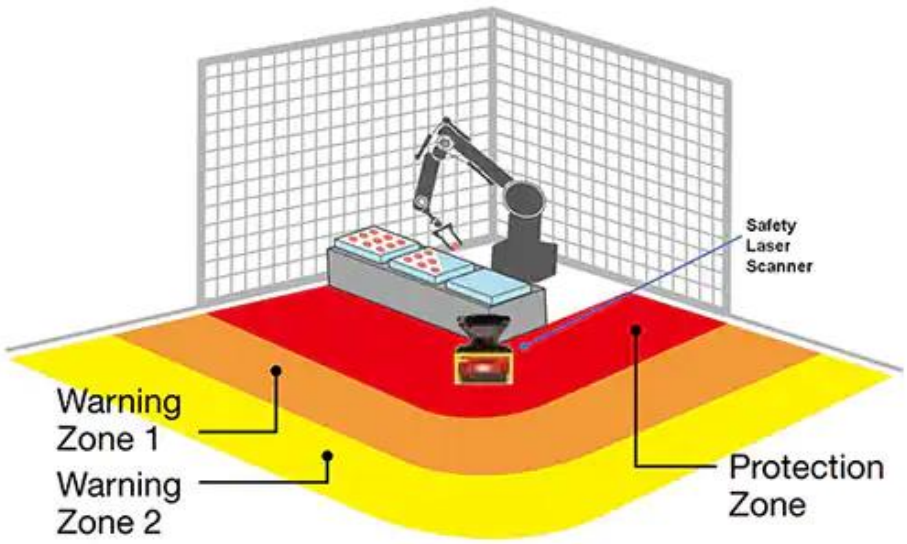


Figure 4.4-2: Safety laser scanners and its zones

The Pilz safety laser scanner was used to signal the robot for the dangers that might happen. The defined safety zone for the AUBO i10 robot has two zones. The warning zone signals the robot to pause the operation until the obstacle goes away from the danger zone. The distance defined for the warning zone is infinity to a minimum distance of two meters. The circuit diagram for safeguard is shown in the Figure 4.4-3

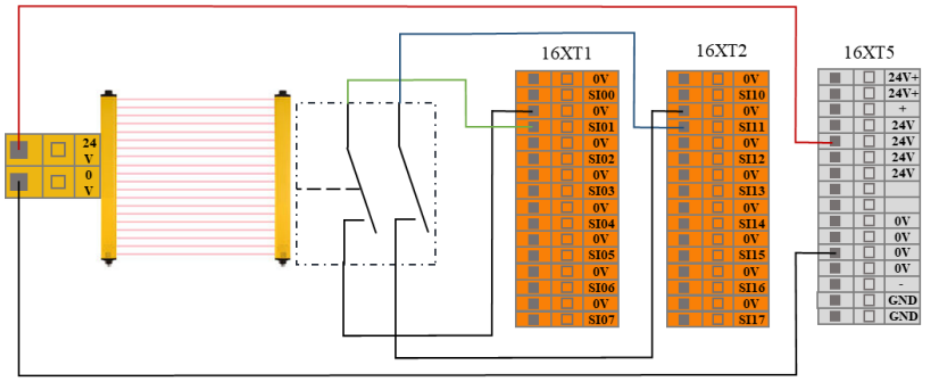


Figure 4.4-3: Safeguard wiring that causes a pause in operation

The circuit should always be connected. It means that the input SI01 must be connected to the input 0V, as seen in the picture. If the circuit is disconnected, it signals to the robot that the safeguard is activated. In this case, the Pilz or any other safety device's output should trigger a relay. A relay is needed because the output of the safety devices is usually a +24v of voltage as the trigger. On the other hand, the AUBO i10 does not handle +24v triggers but the circuit shown in Figure 4.4-3. As long as the relay is not triggered, the circuit is connected.

If the obstacle reaches the protection zone, which is equal to a maximum of 2 meters to a minimum of 1.3 meters, the safety laser scanner sends a signal from another output. in case of activating the protection zone, the emergency stop of the robot is activated, and robot is turned off until an operator goes on-site and handle the problem. Figure 4.4-4 shows the circuit to handle emergency stops.

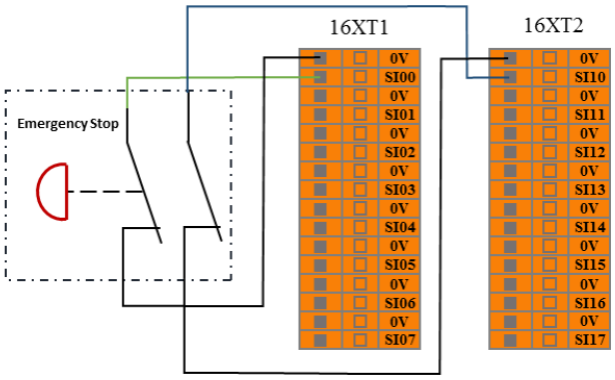


Figure 4.4-4: Emergency stop wiring of AUBO i10

Like the safeguard circuit, a relay is also needed for an emergency stop. Meanwhile, the safety signal is sent to the robot directly; the same signal is also sent to the

integrated system. The RevPi DIO module has the ability to receive the +24v signals. For instance, the same +24v of safeguard signal that triggers the relay is also triggered the digital input of the RevPi DIO. In the Figure 4.4-5 you can see how the related node of inputs receive the signal.

The circuit above requires a normally-closed relay. We are in a normal state by default, and everything is connected. Once a problem is detected, the signal activates the relay, and the circuit is cut. To demonstrate on relays more effectively, normally open relays are set to the open position by default, which means that there is no contact between the circuits when they are not in use. When power is applied, an electromagnet contacts the first circuit to the second, closing it and allowing power to flow through. When the electricity is turned off to stop the flow, the circuit reopens. Unless otherwise specified, normally closed relays default to the closed position, which means that the circuit is closed. Applying power draws the first circuit away from the second circuit, effectively shutting it down.

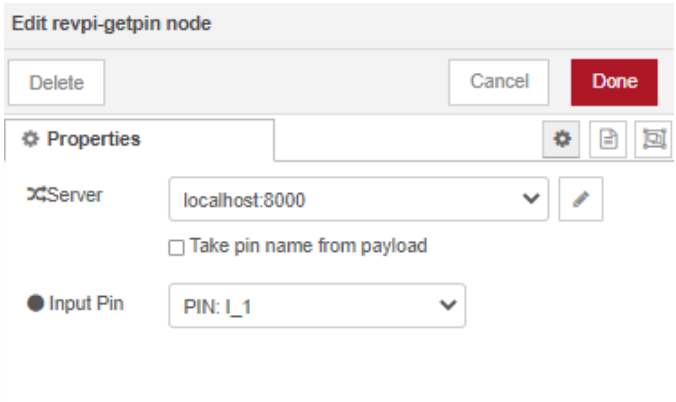


Figure 4.4-5: Properties of RevPi DIO getpin on Node-RED

4.4.2 Safety handler of the integrated system

The overall structure of the safety system was demonstrated. In this section, the implemented safety handler of the integrated system will be elaborated. We know that in this work we have tried to implement a safety system that is working simultaneously in parallel with the integrated system.

When an emergency event arises, the signal is delivered to both the field device directly and to the integrated system. The idea is that the integrated system should be aware of the circumstance to commit further instructions.

A proximity sensor and two inputs from the laser scanner are introduced to the system as three sources of safety signals. Emergency signals are defined by ES+name

of the source, which in this case, the proximity sensor is ESUS and the laser scanner triggers are ESPLC1 and ESPLC2. The ESGlobal flag states whether the system is sensing an emergency signal or not. This flag is set automatically by signals received from field devices or safety devices. The EM flag shows the state of the system, whether it is in emergency mode or not. Putting the system into emergency mode is done by ESGlobal or manually by a human operator. Turning off the EM is only possible by a human operator by checking the condition that the ESGlobal is turned off or not. For instance, if an operator wants to turn off the EM, there might be a possibility that the system is still sensing a safety signal. In this regard, safety was developed based on the following approach:

- The laser scanner shows a close obstacle in the safety zone.
- The global variable ESPLC1 is set to 1.
- The global variable ESGlobal is set to 1.
- The global variable EM is set to 1.
- The emergency mode register of field devices is set to 1
    - The emergency might happen to one of the field devices but, the rest of the field devices should be aware of the condition.
- The laser scanner shows that the zone is safe.
- The global variable ESPLC1 is set to 0.
- The global variable ESGlobal is set 0 to if ESPLC1 and ESPLC2 are also 0.
- The EM variable remain in 1 until further command from human operator

The above procedure is implemented in the integrated system when it receives a signal from safety devices. As it was stated above, the same signal is also sent to the field device as well. Each field device handles the safety signals according to its built-in instructions. This instruction stops the current process and remains in the situation to get further commands. This part of the development would be different based on each specific field device which the case of AUBO i10 robot was explained in section 4.4.1

4.4.3 Software, unauthorized access

Modbus protocol has lacked security with no confidentiality and data integrity. The rise of insecure network protocols such as Modbus is often accompanied by the great dangers of application in real-world systems, especially critical ones. Using Wireshark and Scapy to exploit weaknesses in the Modbus over TCP/IP can happen by third parties if we do not find any solution to secure the protocol. [19]

A variety of works have been done to secure Modbus, like Modbus TLS, and most of them are successful works. We did not try to implement a secure Modbus or secure it

in a new way in this project. We have implemented the system in a way that the network of field devices is wholly separated from the Internet. The field devices are connected to the integrated system, and any other requests from undefined sources will be dropped. The only possibility to access the system would be physical access to the equipment. Steps taken to avoid any unauthorized access are elaborated in the following. [19]

4.4.4 Secured integrated system

The integrated system is a two-sided platform. On one side, it interacts with the field devices while the other side is connected to the Internet. The devices in the field have no direct connection to the Internet. Consequently, if anyone wants access to the field devices, she should get access to the integrated system. The integrated system is fully exposed to the Internet, while a firewall in between could be helpful to restrict the access.

We have mentioned that there are two ways to access the HMI of the integrated system. We establish a TCP connection or connect to the MQTT broker. The MQTT protocol provides a reliable and secure connection. The security is applied in different levels.

- Network Layer

Using a physically secure network or VPN for all communication between customers and brokers is one technique to establish a safe and trustworthy connection. This method is appropriate for gateway applications in which the gateway is connected to devices on the one hand and the broker through VPN on the other.

- Transport layer

TLS/SSL is used a lot for transport encryption where security is the primary goal. This is a safe and reliable way to ensure that data is not read while it is being sent, and it also lets both parties use client certificates to prove their identities. It is possible to use TLS on devices that are not allowed to use. In a separate piece, we go into this more in-depth.

- Application layer

At the transport level, communication is encrypted and identities are verified. MQTT provides a client identity and username/password credentials for application-level device authentication. These features are included in the protocol itself. The broker

implementation is unique in that it defines the authorization or control that each device is permitted to exercise. Additionally, payload encryption can be used to secure transmitted data at the application level. [20]

To ensure application and transmission layer security, the MQTT broker is configured to use TLS encryption and each identity is authenticated using a username and password.

Depending on the network's security policies, it may be possible to connect to the network via a VPN to add another layer of security to the network layer.

5. Experimental results

The result of the project is a web-based HMI that the operator can monitor and control the field devices. The current version of the integrated system is implemented to translate Modbus protocol to MQTT and vice versa. The requirements of the thesis topic are met, and the implemented system is working perfectly. The latest version of the HMI can be seen in the figure below. We have only one field device in this version, the AUBO i10 robot. The robot's goal is to pick up and move a sample to the measurement chamber, take it out of the chamber, and put it in the initial position. There are five samples in general, while two different movements are defined for each sample. Move 1 to pick up the sample and put it inside the chamber, while move 2 takes out the sample from the chamber and puts it in on initial position.

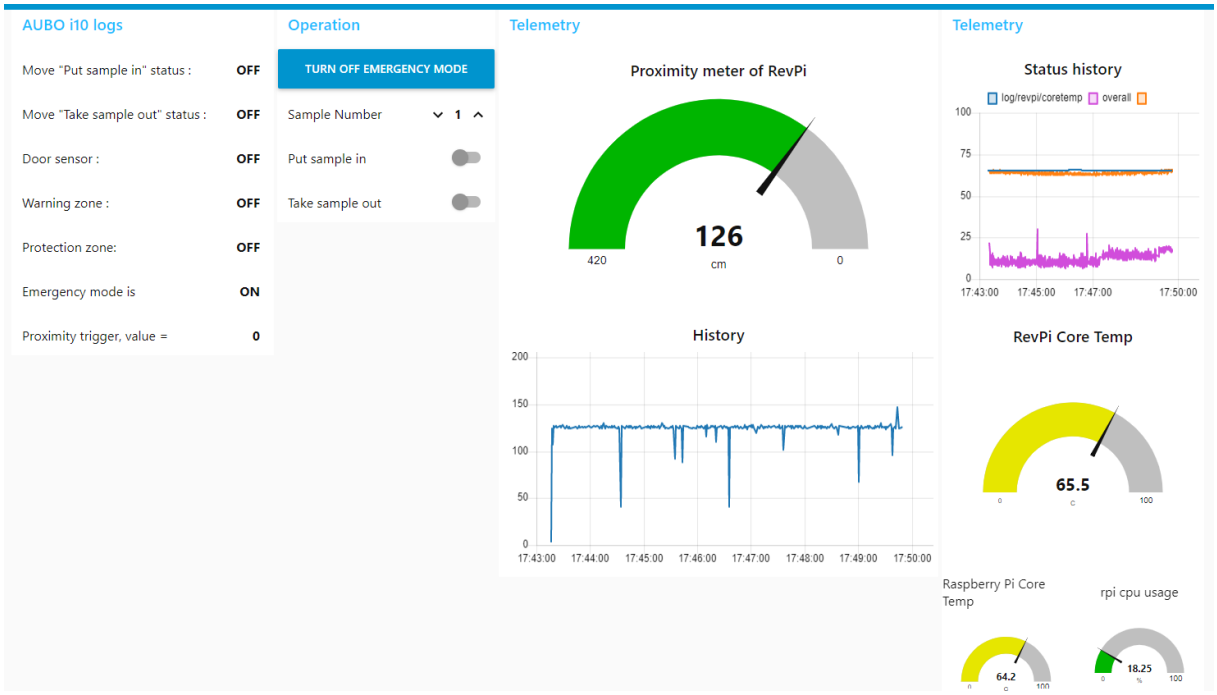


Figure 4.4-1: The HMI of integrated system

The integrated system carries the duty of remote communication to control the robot. We know that the AUBO i10 takes advantage of Modbus TCP to communicate. We have used Modbus communication to connect to the robot and control it to move the samples. Two variables have been used to refer to the type of movement and the

number of samples. Once the robot completes the operation, it sets the variables to a default value of 0. Variables are the registers of the Modbus server, in fact.

The above-mentioned scenario is a real scenario tested in the RFID laboratory. Figure 4.3-3 shows the RFID laboratory.

5.1 Stress test

A real environment would always be a challenge for a newly developed system. This section defines three different scenarios to put the integrated system into a challenge to see how it works.

5.1.1 Scenario 1

AUBO i10 has provided a simulation environment to test. The simulator is a virtual machine similar to the one you can use to program the robot. Worth mentioning that the robot is taking advantage of the Ubuntu distribution of the Linux operating system. An AUBUPRE application is installed in which you are able to program the robot.

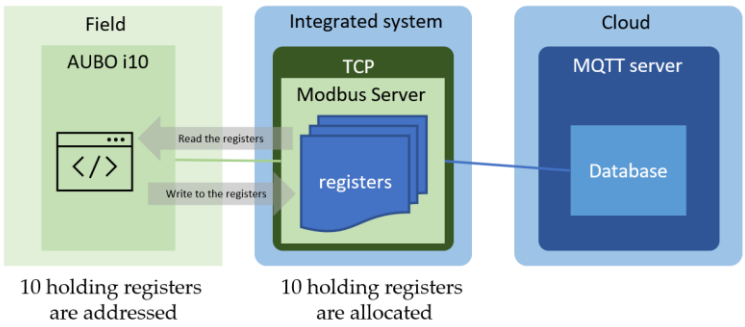
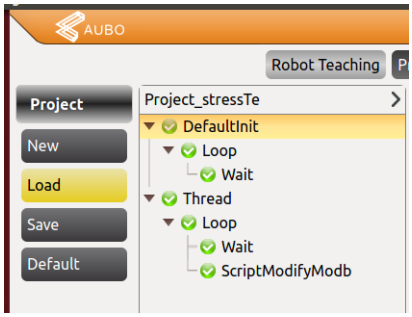


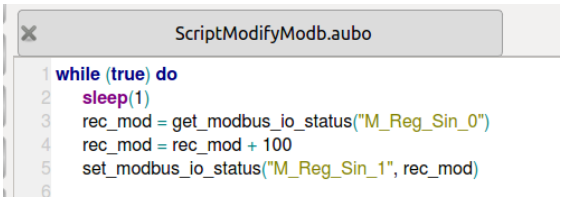
Figure 5.1-1: Scenario 1

The company provides the same operating system with the AUBUPRE application as a virtual machine, making it easier to do a virtual experiment. To proceed with the stress test, we exploit the simulator to read and write the registers of the Modbus server.

The machine is reading 5 Modbus server registers, modifying the value, and writing in the next register in line. The goal is to simulate a situation where the robot actively reads several external variables and performs an action based on them while the result is also sent back to the integrated system.



(a) Main program



(b) A piece of script

Figure 5.1-2: Implemented AUBO i10 program used to execute testing scenarios

The transmitted and received data is sent to the remote HMI using MQTT in the next step. In this scenario, 10 holding registers are modified while all of them are sent to the HMI by one publish commands to be stored in a CSV file.

The holding register number 0 to 9 are allocated to write in the values. The values are decrement counters that you can enter the number that you want to start from. The value are decremented each 2 seconds and are written to registers number 0,2,4,6 and

	reg0	reg1	reg2	reg3	reg4	reg5	reg6	reg7	reg8	reg9
2	0	0	0	0	0	0	0	0	0	0
3	100	200	100	200	100	300	100	400	100	500
4	99	199	99	300	99	400	99	500	99	600
5	98	198	98	299	98	399	98	499	98	599
6	97	197	97	298	97	398	97	498	97	598
7	96	196	96	297	96	397	96	497	96	597
8	95	195	95	296	95	396	95	496	95	596
9	94	194	94	295	94	395	94	495	94	595
10	93	193	93	294	93	394	93	494	93	594
11	92	192	92	293	92	393	92	493	92	593
12	91	191	91	292	91	392	91	492	91	592
13	90	190	90	291	90	391	90	491	90	591
14	89	189	89	290	89	390	89	490	89	590
15	88	188	88	289	88	389	88	489	88	589
16	87	187	87	288	87	388	87	488	87	588
17	86	186	86	287	86	387	86	487	86	587
18	85	185	85	286	85	386	85	486	85	586
19	84	184	84	285	84	385	84	485	84	585
20	83	183	83	284	83	384	83	484	83	584

Figure 5.1-3: Result of stress test

8. The AUBO i10 is running an script that reads the mentioned registers and adds a constant to write them in the registers number 1,3,5,7 and 9 right away.

The data are read and sent to the remote client using MQTT. The result should be a continuous series of numbers meaning that we did not lose any data during translation and transmission. The test run was done five times with five different

initial numbers to count down. In Figure 5.1-3 you can see the result of run 4, counting down from 100 to 1. The picture shows the first 20 results.

5.1.2 Scenario 2

The result of the last scenario showed that the system is working fine with one field device. In this scenario, we put a step further and increase the number of field devices that are connected to the integrated system through Modbus. The number of AUBO i10 VMs increased to 4 VMs. Each of them is connected to the system as in the first scenario.

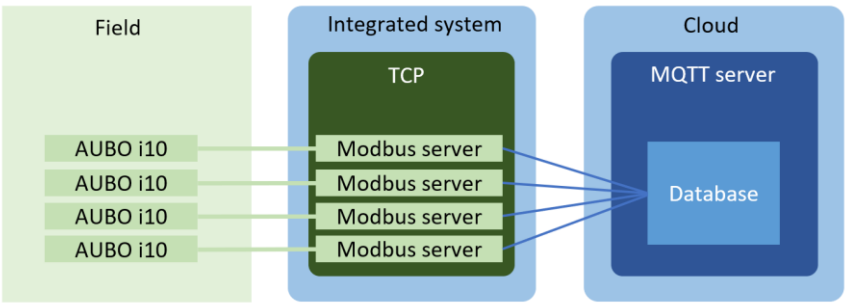


Figure 5.1-4: Scenario 2

This test aims to challenge the integrated system to see whether it can handle more than one field device or not. Each VM is connected to only one Modbus server. The servers are distinguished by their port number. The rest of the configurations are similar to the previous scenario.

Since the RevPi Core3 has enough hardware resources, the test result was the same as the previous test as it was expected. As long as we have enough hardware resources, we can increase the number of field devices managed by one single integrated system.

5.1.3 Scenario 3

In a real environment, it rarely happens that a field device reads, modify or write a value every 2 seconds like the two previous scenarios. The third scenario is more realistic in such a way that we try to implement a situation in which three AUBO i10 robots are operating simultaneously based on the value of the Modbus registers. The Modbus server, like before, is implemented in the integrated system side. The goal of this test is to simulate a condition that field devices should be aware of each other while each one of them can be controlled separately by the HMI.

The operator sends the command to the robot. Meanwhile, she can decide to involve the other robots in operation or not. The following algorithm shows the instructions while you can see the architecture of the scenario in the Figure 5.1-5.

Algorithm 3 algorithm followed by robots to interact with each other

```
1:  initial instructions
2:  wait for the comand
3:  if move ==1 then
4:      initiate operation robot1
5:      set job is done
6:  end if
7:  if move == 2 then
8:      initiate operation
9:      if robot2.activity == true then
A:          initiate operation of robot2
B:          if robot3.activity == true then
C:              initiace operation robot3
D:              set job is done robot3
E:          end if
F:          set job is done robot1
10:         set job is done robot2
11:     end if
12:     move the robot to zero possition
13: end if
14: go to line 2
```

Each AUBO i10 has a program to operate the robot. The operations are done based on conditions that the operator defines. When an operation is done, the robot returns feedback whether the job is done or not. The duty of holding registers 1 and 2 is to save the robot's status while the coil registers 0 and 1 ask the robots 2 and 3 to start their job or not.

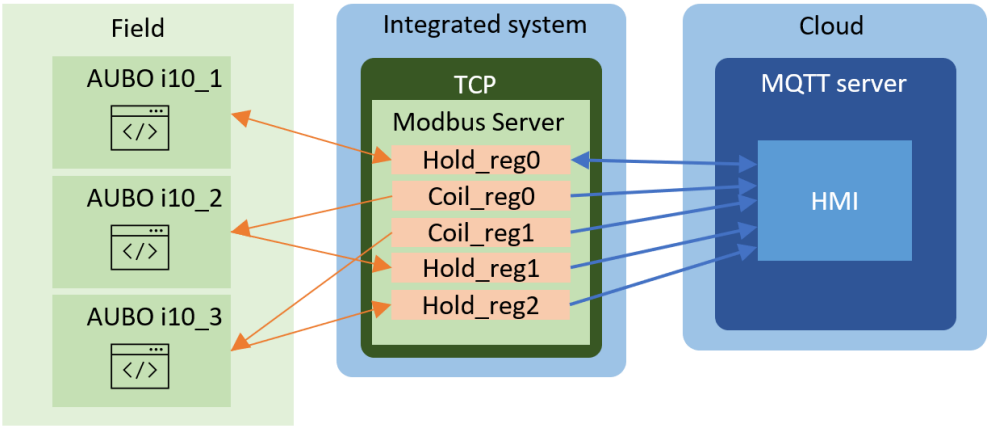


Figure 5.1-5: Scenario 3

The third scenario's expectation was met, and the three robots were working together as a collaborative process. No problems were encountered when using the shared registers of the Modbus server, and the procedures were carried out according to the sequence defined in Algorithm 3.

So far, the experimental results have indicated that the integrated system is performing satisfactorily; however, there will be challenges and bugs that may arise during the system's long run operations.

6. Conclusion and future development

In this work, we have explored the world of IIOT to see how companies are dealing with field devices and how they can stick to the concept of Industry 4.0.

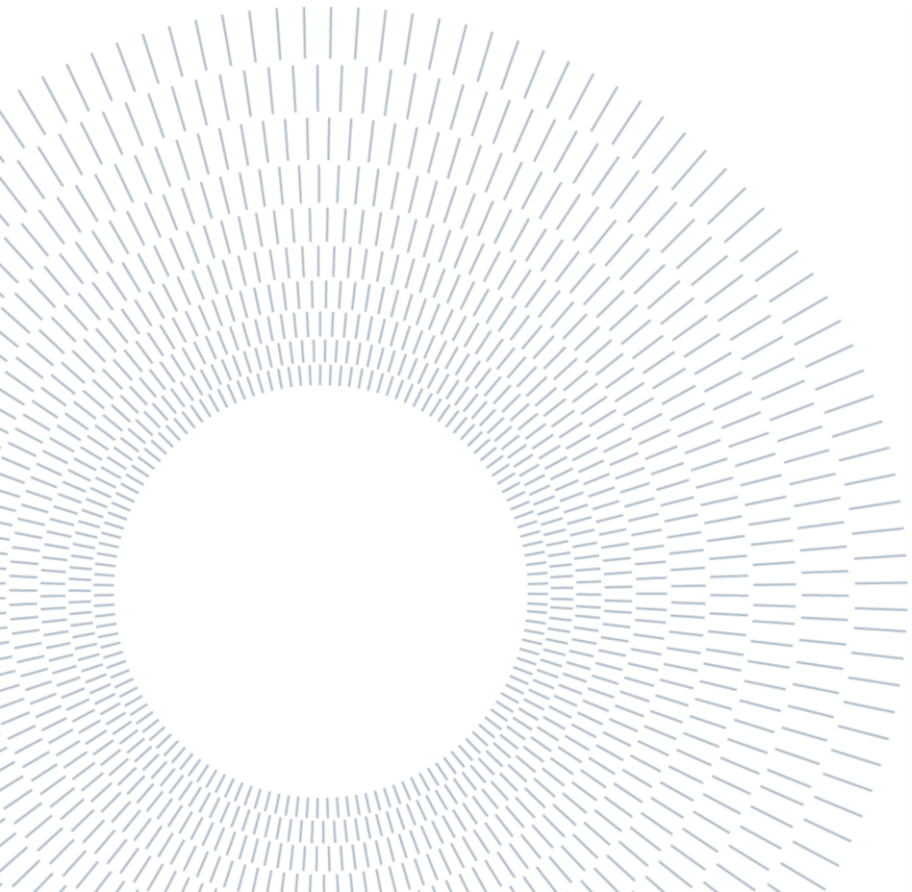
We have started with a brief demonstration on the hardware and software tools needed to implement the project to give the reader have a better mindset to understand the world of the Internet of Things in industry.

In the next step, we tried to show how the system was implemented, why a particular system architecture was chosen, why specific hardware is more suitable than the others and finally, how we can stick all the components to have an integrated system that satisfies the needs. Also, we learnt that security is a crucial and inseparable part of the industry that should put more effort into it.

Although the safety chapter was not very detailed since the area is too wide to be covered in this work, it would be a good topic for future work on maintaining safety and security when we expose a non-secure protocol to the public network.

Last but not least, we came to a conclusion with positive and satisfying experimental results. Three scenarios were defined and implemented to test the system in a virtual environment that satisfied all the expectations. Also, a real scenario was tested in the laboratory, which was successful.

At this point, there are still many vacancies for future developments. How can we develop the system in a way that supports other protocols? What would be the best solution to secure an insecure protocol when exposed to public networks? Would Image processing be an excellent approach to gain safety? By developing platforms to operate field devices remotely, can we have a Laboratory as a Service?



Bibliography

[1] COPADATA, "What is Iot vs IIot?," [Online]. Available: https://www.copadata.com/en/product/platform-editorial-content/what-is-the-iot-and-iiot/.

[2] V. Georgieva, 2016. [Online].

[3] Utthunga, "All You Need to Know About Industry Protocols," [Online]. Available: https://utthunga.com/blogs/all-you-need-to-know-about-industry-protocols/.

[4] P. D. S. Steinhorst, "https://www.ei.tum.de/esi/forschung/interoperability-of-industrial-iot-systems/," [Online].

[5] R. pi, "revolution-pi-series," [Online]. Available: https://revolutionpi.com/revolution-pi-series/.

[6] AUBO, "public/iproduct4," [Online]. Available: https://www.aubo-cobot.com/public/iproduct4.

[7] T. briefs, "Collaborative Robots vs. Traditional Robots: What's Right for Your Manufacturing Applications," [Online]. Available: https://www.techbriefs.com/component/content/article/tb/stories/blog/36314.

[8] T. M. Organization, "About Modbus Organization," [Online]. Available: https://modbus.org/.

[9] e. t. solutions, "MODBUS PROTOCOL," [Online]. Available: https://embeddedtechsolutions.com/modbus-protocol/.

[10] S. Modbus. [Online]. Available: https://www.simplymodbus.ca/.

[11] D. Meador, "Semaphores in Operating System," 2018. [Online]. Available: https://www.tutorialspoint.com/semaphores-in-operating-system.

[12] S. Yang, Y. Zhong, D. Feng, R. Y. M. Li, X.-F. Shao and W. Liu, "Robot application and occupational injuries: Are robots necessarily safer?," *Safety Science,* vol. 147, 2022.

[13] V. Murashov, F. Hearl and J. Howard, "Working Safely with Robot Workers: Recommendations for the New Workplace," *Journal of Occupational and Environmental Hygiene,* p. D61–D71, 2016.

[14] A. Zingman, G. S. Earnest, B. D. Lowe and C. M. Branche, "Exoskeletons in Construction: Will they reduce or create hazards?," *NIOSH Science Blog. U.S. National Institute for Occupational Safety and Health,* 2017.

[15] M. Vasic and A. Billard, "Safety issues in human-robot interactions," *IEEE International Conference on Robotics and Automation,* p. 197–204, 2013.

[16] J. Howard, V. Murashov and C. M. Branche, "Unmanned aerial vehicles in construction and worker safety," *American Journal of Industrial Medicine,* pp. 3-10, 2018.

[17] T. N. I. f. O. S. a. Health, "Preventing the Injury of Workers by Robots," [Online]. Available: https://www.cdc.gov/niosh/docs/85-103/.

[18] O. S. a. H. Administration, "Occupational Safety and Health Administration," [Online]. Available: https://www.osha.gov/otm#3.

[19] C. G. T. &. Parian and S. Bhatia, "Fooling the master: Exploiting weaknesses in the Modbus protocol," *Procedia Computer Science,* vol. 171, p. 2453–2458, 2020.

[20] HiveMQ, "HiveMQ - MQTT Security Fundamentals," [Online]. Available: https://www.hivemq.com/blog/introducing-the-mqtt-security-fundamentals/.

A. Appendix A

A.1 Demonstration on Nodered-modbus-contrib nodes

- Modbus-Response
  Node to show response or response length from second output of Modbus Read/Write/Getter nodes in status.
- Modbus-Read
  Connects to a Modbus TCP or serial to read registers/coils values with a given polling rate. Function codes currently supported include:
  - FC 1: Read Coil Status
  - FC 2: Read Input Status
  - FC 3: Read Holding Registers
  - FC 4: Read Input Registers
- Modbus-Getter
  Modbus TCP/Serial node with triggered read function by input. Connects to a Modbus TCP or serial to read coils/inputs/registers at the rate of the incoming message. Function codes currently supported include:
  - FC 1: Read Coil Status
  - FC 2: Read Input Status
  - FC 3: Read Holding Registers
  - FC 4: Read Input Registers
- Modbus-Write
  Modbus TCP/Serial node triggered with msg.payload to write. Connects to a Modbus TCP or serial to write coils/registers at each incoming msg. Function codes currently supported include:
  - FC 5: Force Single Coil

  - FC 6: Preset Single Register
  - FC 15: Force Multiple Coils
  - FC 16: Preset Multiple Registers

- Modbus-Flex-Write
  Modbus TCP flexible input triggered write node with connection input parameters. Connects to a Modbus TCP or serial to write coils/registers on each incoming msg.
  Function codes currently supported include:
  - FC 5: Force Single Coil
  - FC 6: Preset Single Register
  - FC 15: Force Multiple Coils
  - FC 16: Preset Multiple Registers

- Modbus Server
  Node to provide a Modbus TCP server based on node-modbus (jsmodbus) for testing. On injecting the server sends the Buffers to the separate outputs You can use the Modbus write nodes (FC) to write data to the server buffers.
  You can use the Modbus read nodes (FC) to read data from the server buffers.
  - Output 1: holding Buffer, type, msg
  - Output 2: coils Buffer, type, msg
  - Output 3: input Buffer, type, msg
  - Output 4: discrete Buffer, type, msg

  About the Input, on injecting a special payload, you can write directly to any register. This should only be used if you want to simulate a Modbus client. msg.payload = { 'value': msg.payload, 'register': 'holding', 'address': 1 , 'disableMsgOutput' : 0 }; return msg;
  The value could also be a list of UInt8 numbers and they will be written to the buffer. Valid registers are:
  - holding
  - coils
  - input
  - discrete

  Set disableMsgOutput if you want to disable the Server outputs when injecting.

- Modbus-Flex-Server

Node to provide a flexible Modbus TCP server based on modbus-serial for testing. The modbus-serial package allows to configure/inject code to handle modbus requests. With that you can build you personal Modbus server. The function frame is fix - just the body of functions is flexible. On injecting the server sends the Buffers to the separate outputs. You can use the Modbus write nodes (FC) to write data to the server buffers. You can use the Modbus read nodes (FC) to read data from the server buffers.

- Output 1: holding Buffer, type, msg
- Output 2: coils Buffer, type, msg
- Output 3: input Buffer, type, msg
- Output 4: discrete inputs Buffer, type, msg
- Input:
  On injecting a special payload, you can write directly to any register. This should only be used if you want to simulate a Modbus client. msg.payload = { 'value': msg.payload, 'register': 'holding', 'address': 1 , 'disableMsgOutput' : 0 }; return msg;
  The value could also be a list of UInt8 numbers and they will be written to the buffer.

Valid registers are:

- holding
- coils
- input
- discrete

Set disableMsgOutput if you want to disable the Flex Server outputs when injecting.

- Modbus-Queue-Info
  A queue is set per unit - setup the unit-id to get the right information. Modbus TCP/Serial queue information node. Unit-Id (0..255 tcp | 0..247 serial)
  Unit-Id 0 is for broadcasting Modbus messages.

- Modbus-Flex-Connector
  Modbus Flex Connector is a node for flexible input triggers to reconnect with new connection parameters.

- Modbus-Response-Filter

The Response Filter is to work with an IO-File to filter values out of an IO-Payload by name. Deploy first to use the lookup button! You need a working IO-File-Config for the lookup.

- Modbus-Flex-Sequencer
  Modbus TCP/Serial flexible input triggered auto-sequence read node with connection input parameters. Connects to a Modbus TPC or serial to read coils/inputs/registers automatically in sequence. Use any input to trigger the request but you can override the list using:

  msg.sequences
  Function codes (1:4) currently supported include:
  - FC 1: Read Coil Status
  - FC 2: Read Input Status
  - FC 3: Read Holding Registers
  - FC 4: Read Input Registers

  Input parameter for connecting Modbus

  - (mixed) fc ('FC1', 'FC2', 'FC3', 'FC4' - Or - 1:4) - Function Code
  - (integer) unitid (0..255 tcp | 0..247 serial) - Unit-ID
  - (integer) address (0:65535) - The starting address
  - (integer) quantity (1:65535) - The quantity of coils/inputs/registers to be read from the starting address

  Output 1: data Array (PDU), modbus response Buffer, input message
  Output 2: modbus response Buffer, data Array (PDU), input messag

List of Figures

Figure 2.1-1: Revpi Core module 7
Figure 2.1-2: Revpi IO modules 8
Figure 2.1-3: PiCtory user interface 9
Figure 2.2-1: AUBOi10 robot 10
Figure 2.3-1: Modbus RTU 11
Figure 2.3-2: Modbus TCP/IP 11
Figure 2.3-3: Modbus data transaction 13
Figure 2.3-4: This simple network example for Modbus TCP/serial. 15
Figure 2.4-1: Node-RED environment 16
Figure 3.3-1: The overall system architecture 21
Figure 3.3-2: More detail in the architecture of the project 22
Figure 3.4-1: Modbus configuration menu on AUBO i10 23
Figure 3.4-2: Configuration page of "modbus-server" node on Node-RED 24
Figure 3.4-3: Modbus server and client 25
Figure 3.4-4: Header of Modbus RTU and Modbus TCP 28
Figure 3.5-1: Properties of “modbus-write” node 30
Figure 4.3-1: Safety zone of an arm robot 37
Figure 4.3-2: The workspace of the manipulator 38
Figure 4.3-3: Working environment of AUBO i10 39
Figure 4.4-1: Reduced mode input connection triggered by door sensor 42

Figure 4.4-2: Safety laser scanners and its zones 42
Figure 4.4-3: Safeguard wiring that causes a pause in operation 43
Figure 4.4-4: Emergency stop wiring of AUBO i10 43
Figure 4.4-5: Properties of RevPi DIO getpin on Node-RED 44
Figure 4.4-1: The HMI of integrated system 49
Figure 5.1-1: Scenario 1 50
Figure 5.1-2: Implemented AUBO i10 program used to execute testing scenarios 51
Figure 5.1-3: Result of stress test 51
Figure 5.1-4: Scenario 2 52
Figure 5.1-5: Scenario 3 54

List of Tables

Table 3-1: Standard addresses in Modbus 26
Table 3-2: Function codes of Modbus protocol 26
Table 4-1: Emergency I/O interface of AUBO i10 robot 40

Acknowledgements

This thesis is a collaborative work with the Hochschule für Technik Stuttgart as a part of the DigiLab4U project.

Throughout the writing of the thesis, I have received a great deal of support and assistance. I would like to acknowledge and give my warmest thanks to my supervisors, Dr Alessandro Enrico Cesare Redondi from Politecnico di Milano and Dr Dieter Uckelmann from HFT Stuttgart, who made this work possible. Their guidance and advice carried me through all the stages of writing my project.

I would like to acknowledge and be grateful to work with a wonderful colleague, Hadi Adineh, who provided so much help in my research. At the same time, his advice was valuable to me during the work. I would also like to thank the DigiLab4U team members at HFT Stuttgart for their valuable support and making my moments of stay in the team be enjoyable.

In addition, I would like to thank my parents for their wise counsel and sympathetic ear. You are always there for me. Finally, I could not have completed this dissertation without the support of my friends, specially Erfan Abbasi, who introduced me to the research team and provided stimulating discussions and happy distractions to rest my mind outside of my research.