Expertise
and insight
for the future

Patrik Asikainen

Instrumentation of Distillation Column Using Open-Source Platforms

Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Programme in Electronics

Bachelor's Thesis

03 May 2021

Author Title Number of Pages Date	Patrik Asikainen Instrumentation of Distillation Column Using Open-Source Platforms 21 pages + 2 appendices 03 May 2021
Degree	Bachelor of Engineering
Degree Programme	Degree programme in Electronics
Professional Major	Electronics
Instructors	Erkki Räsänen, Principal Lecturer Timo Seuranen, Senior Lecturer

Cyber-physical systems, many claims, are the solution to the never-ending problem of increasing production efficiency in automated industries, and like with many other new technologies, it comes with its own set of challenges that need to be overcome before businesses are willing to make the jump.

The objective of this thesis was to create a new remote control and monitoring system for a distillation column using open-source technologies, and in doing so, evaluate their viability in tackling some of the problems of implementing CPS in the industry.

An initial prototype was designed using a Controllino MEGA as the PLC with DS18B20 temperature sensors placed at key points in the distillation column for a good temperature profile of the distillation process. MQTT is the chosen communications protocol and Node-Red the tool used to create a functioning SCADA system.

The project reached a satisfying cutting point with the temperature monitoring system implemented. It can showcase the potential of the technologies used in the industry; however further work needs to be done to complete the control system before a more conclusive verdict can be achieved.

Keywords	Cyber-physical system, distillation column, instrumentation, PLC, IoT, open source.

Contents

Appendix 1. Code used to retrieve IP data.

Appendix 2. Code used in the project.

List of Abbreviations

CPS	Cyber-Physical Systems. Integration of embedded computers and network monitor and control to physical processes, which affect computations through a feedback loop.
HMI	Human-Machine Interface. Refers to a feature that enables humans to easily engage and interact with machines through an interface.
I^2C	Inter-Integrated Circuit. It is a synchronous, multi-master, multi-slave, packet switched, single-ended, serial communication bus which allows for intra-board communications with lower-speed peripheral integrated circuits.
IDE	Integrated Development Environment. Software application that allows programmers to write and edit code. Generally, it also has tools to aid in debugging.
IoT	Internet of Things. A network of physical objects that, with the help of sensors, software, and other technologies, can connect and exchange data with other devices and systems over the internet.
IP	Internet Protocol. Unique address used to identify a device connected to the internet or a local network.
JSON	JavaScript Object Notation. Lightweight data interchange format that it is easy for humans to read and easy for machines to parse through, based on the JavaScript programming language.
MAC	Media Access Control. Physical address of a device which is unique in nature and allows it to be identified when connected to a network.
MES	Manufacturing Execution System. Information system that connects, monitors and controls complex manufacturing systems and data flows on the factory floor.

MQTT	Message Queuing Telemetry Transport. A network protocol used for communication between devices. It uses a publish-subscribe messaging pattern and it is designed for connections where a small code footprint is required, or the network bandwidth is limited.
PID	Proportional-Integral-Derivative. Control loop mechanism used in control systems to control process variables by continuously calculating an error value for the desired set point and applying a correction based on proportional, integral, and derivative terms.
PLC	Programmable Logic Controller. Industrial digital computer that has been adapted for control of manufacturing processes.
SCADA	Supervisory Control and Data Acquisition. Control system used to monitor and control devices on the production floor.
SPI	Serial Peripheral Interface. Synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers.
UART	Universal Asynchronous Receiver-Transmitter. Computer hardware device for asynchronous serial communication with configurable transmission speeds.

1 Introduction

The manufacturing industry is in a constant state of evolution and innovation with the aim of increasing production efficiency and profit. With the rise of the Internet of Things (IoT) technologies and more interconnectable devices, initiatives such as Industry 4.0 [1] and Made in China 2025 [2] were launched over the last couple of years. This aims to further digitalize companies within the manufacturing industry and connect several aspects of workflow into a single digitalized environment, often referred to as smart manufacturing.

Cyber-physical systems (CPS) are at the core of smart manufacturing and may be the replacement for current systems being used, but this comes with its own set of challenges. Lack of standardization, concerns over data ownership and challenges with integrating data from different sources were among the top-cited barriers mentioned by companies as indicated in a study made by McKinsey Digital [3,12]. High initial investment may also play a role in discouraging mid- to small-sized companies from taking the risk of adopting a new system. However, using open-source platforms may act as an alternative to proprietary ones, address some of these concerns, and act as a catalyst towards improvement.

Open source is a model that encourages open collaboration, with code, documentation, and blueprints freely available for public use. This not only allows for greater customizability and modularity within the systems but may also relieve some of the financial strain from implementing such systems when compared to proprietary solutions [4].

This project set out to evaluate the viability of CPS implementation using open-source platforms at the field, control, and supervisory levels of automation. The goal was to refurbish a continuous distillation column with a new control and monitoring system using open-source platforms and use this as the basis for judgment.

2 Distillation Columns

Distillation is the process of separating a liquid mixture into two or more components based on their volatility. Substances with high volatility have lower boiling points and conversely, low volatility indicates higher boiling points. This difference in volatility is taken advantage of in distillation columns to separate a mixture of multiple liquids with similar boiling points through a process called fractional distillation. Through this process, a mixture is heated to the temperature at which one or more of its components will vaporize at a given pressure. However, if the boiling points are similar enough, unwanted components might vaporize alongside the intended product. To avoid this, the vapour is passed through a fractionating column which causes the top of the column to be richer with high volatility components, and the bottom of the column with low volatility components.[5, 1-8.] This process is further visualized in Figure 1.

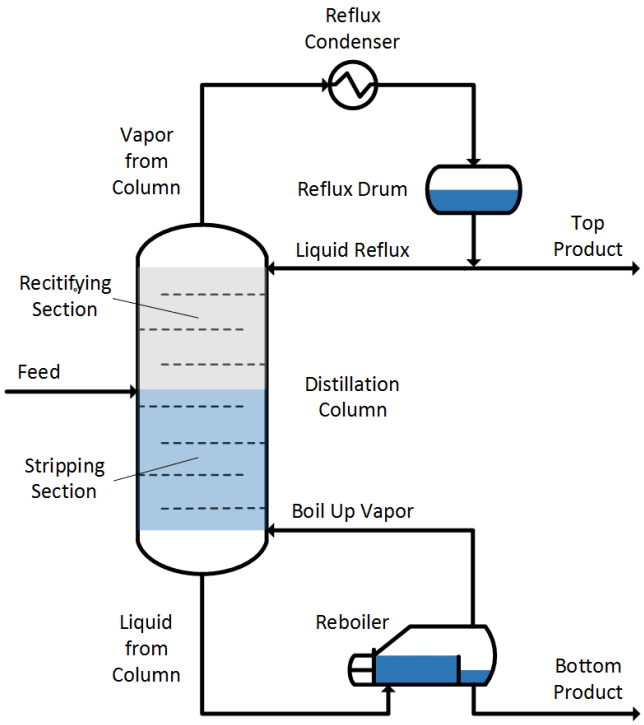

Figure 1. Simple visual of distillation column copied from [6].

It is due to the nature of this process that it becomes vitally important to have a proper temperature profile of the entire column if automation is to be achieved.

3 Cyber-Physical Systems

Cyber-physical systems are systems in which their hardware and software components are deeply intertwined. They integrate sensing, computation, control, and networking into physical infrastructure and can actuate on physical elements and processes based on the data gathered through sensors. The general architecture of such a system consists of programmable logic controllers (PLC) that are connected to sensors, actuators, and motors to monitor and control a system or process. It can also communicate with other devices to send and receive data. [7;8.]

Automation in a typical factory is often represented by a hierarchical pyramid showcasing the automation process at different levels as shown in Figure 2.

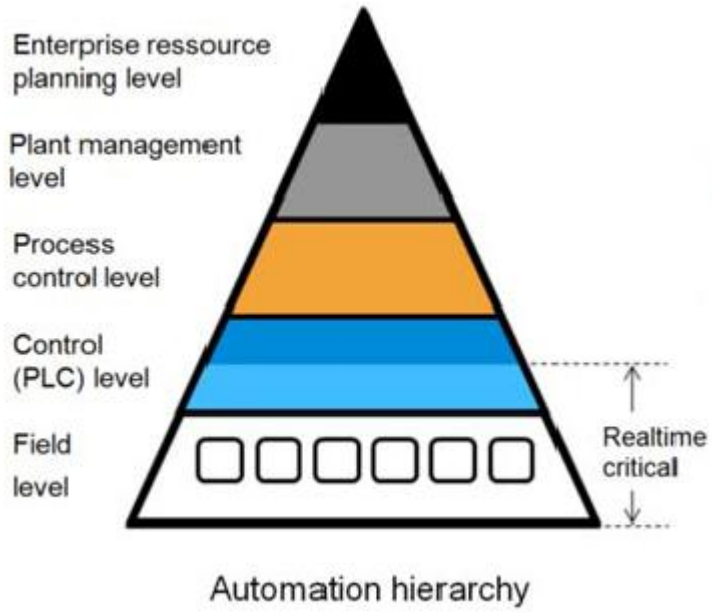

Figure 2. Automation hierarchy pyramid cropped from [8, 4]

- Field level is where sensors and actuators are found. At this level, sensors monitor the physical space around them, and actuators and motors do the physical work. This layer generally works in the order of micro to milliseconds as the components are constantly communicating with the control systems in the next level.

- Control level is where PLC and proportional-integral-derivative (PID) controllers operate. At this level, PLCs use the data from the sensors at the previous level as inputs to create outputs that control the actuators and motors.

- Supervisory level is where supervisory control and data acquisition (SCADA) systems and human-machine interfaces (HMI) are used. At this level, data is collected and monitored through an HMI, as well as stored in databases for future retrieval. SCADA systems are used to control multiple machines at the control level from a single point.

- Planning level is where monitoring systems such as manufacturing execution systems (MES) monitor the entire production life cycle, from the order of raw material to delivery of produced goods, using the data gathered from previous levels. This helps managers make more informed decisions on factory operations and management as events occur.

- Management level is where enterprise integrated management systems are used to oversee the entire operation of the company. It allows top-level management to oversee all levels of business from manufacturing, sales, finance, and payrolls amongst other things. This gives a top-down view of the company's proceedings and improves efficiency and transparency within the company.

CPS-based automation aims to replace the hierarchical pyramid system with a cloud-based network as showcased in Figure 3.

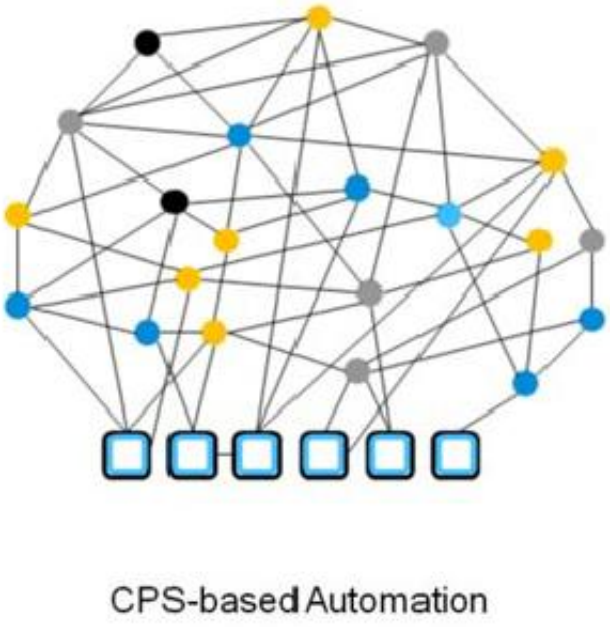

Figure 3. CPS-based automation network cropped from [9, 4]

This type of architecture allows for devices in the field and control level to directly communicate with other layers through a cloud service, instead of cables. This would also support data being exchanged between all levels of the pyramid, not just adjacent layers, allowing for faster and more efficient communications between devices.

Each coloured node in Figure 2 corresponds to its similarly coloured layer in Figure 1 that has had its functionality migrated to the cloud, while the bottom squares represent the mostly unchanged field, partly, the control level. Note in this case, that the control level is divided into two sections. The first part being the physical control which acts upon the field level using PLCs while the second part is migrated to the cloud to offer direct control over it remotely [9].

4 Design and Materials

4.1 Hardware

4.1.1 Controllino MEGA

The Controllino MEGA is an open-source industry-grade PLC made to be fully compatible with Arduino infrastructure. With the aim of providing a reliable open-source alternative to automation and industrial applications. It enables users to control high voltage components with 21 input channels and 24 output channels rated 5-25 Volts, as well as 16 relays that support up to 250 Volts and 16 Amperes. It can be interfaced with industry-standard interfaces such as Serial (Universal Asynchronous Receiver-Transmitter (UART)), RS485, Inter-Integrated Circuit (I^2C) or Serial Peripheral Interface (SPI), It also has network capabilities when connected to a router through Ethernet.

4.1.2 DS18B20 Temperature Sensor

The DS18B20 is a digital thermometer that provides a 9-bit to 12-bit temperature measurement in Celsius over a 1-Wire bus which allows for multiple sensors to communicate through a single data line, making it simple to work with. It can work without the need for an external power supply, as it can derive power directly from the data line (“parasite power”). However, this is not recommended when using a significant number of sensors in a single bus. Each sensor has a unique 64-bit serial code making them simple to identify through a 1-Wire bus and easy to control with a PLC. The package used in this project is the waterproof stainless steel probe sensor with a cable length of 1 meter.

4.2 Software and Architecture

4.2.1 Arduino IDE

The Arduino Integrated Development Environment (IDE) is the official Arduino software used to write code and upload it to an Arduino board. The language is based on C++,

which the IDE then compiles and checks for errors in the program. It also has a built-in Serial monitor for easy debugging.

The Controllino design is based on Arduino architecture so compatibility with its IDE can be easily added. The Arduino IDE also benefits from an extensive collection of official and user-curated libraries to implement into a program.

4.2.2 MQTT

Message Queuing Telemetry Transport (MQTT) is the chosen communications protocol for this project. Due to its lightweight and bandwidth-efficient nature, as well as small code footprint, it is ideal for bi-directional data transfer between PLCs and other systems where memory is limited. It works through a publish-subscribe architecture showcased in Figure 4.

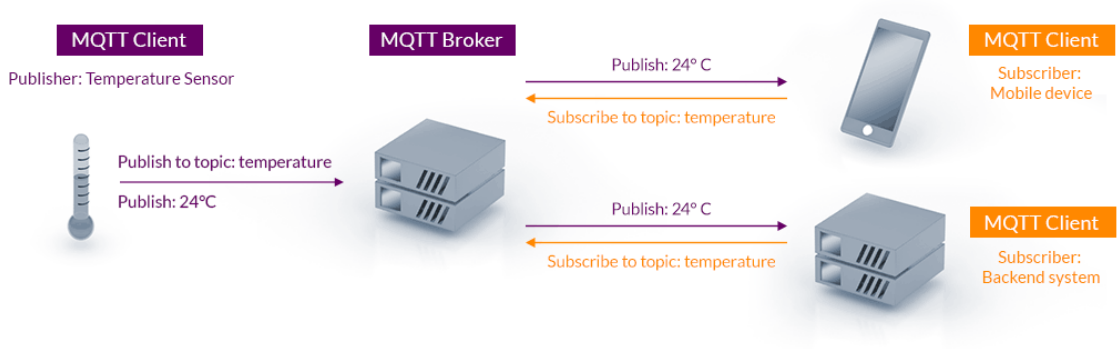

Figure 4. MQTT architecture reprinted from [10].

In this type of architecture, data is published to a topic in the MQTT server, to which multiple devices can subscribe and listen for incoming data. The MQTT server in use for this project is the Eclipse Mosquitto test server. For ease of implementation, port 1883 is used, for which the data passing through remains unencrypted. However, different levels of encryption are available at different ports should the need arise.

4.2.3 Node-RED

Node-RED is an open-source flow-based programming tool for visual programming that is useful for wiring together different hardware devices and software services. It is built on Node.js which uses JavaScript as its main programming language. However, Node-RED is built to prioritize visual programming with minimum coding required for most projects. With a browser-based flow editor, it is easy to wire together the large number of nodes available, both built-in with the software and user-created ones.

Because of its light-weight runtime, it is an ideal tool to use at the edge of the network with low-cost hardware as well as in the cloud, and it is the chosen tool for connecting the devices in use for this project to the database, and an in-built, rudimentary HMI.

4.2.4 MongoDB Atlas

MongoDB Atlas is the chosen database for historian data storage. It is a NoSQL cloud database service that is easy to integrate into a program, easy to navigate through with a web-based interface, and little to no coding required. It was chosen due to its simplicity to set up for testing purposes. However, it can be easily replaced with any other database service.

5 Implementation and Testing

5.1 Design of Control System

The control system is in its prototype stages and consists of a Controllino MEGA as the PLC, connected to a 24 V power supply, with eight DS18B20 Temperature sensors connected to the PLC through a single port to take advantage of its 1-Wire interface features. A picture of the control device can be seen in Figure 5.

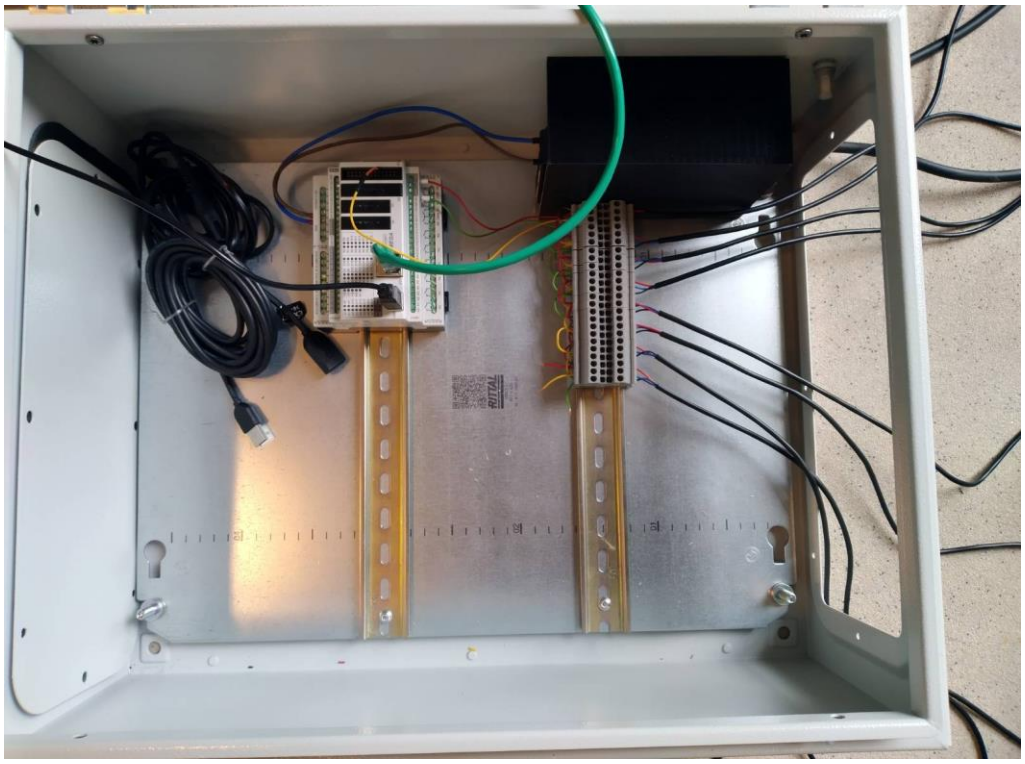

Figure 5. Picture of the control device.

The PLC as well as the connectors for the sensors are held in place through rails. The power supply is covered using a 3D printed cover due to safety concerns over leaving high voltage connections exposed.

Each temperature sensor is made up of three connector pins: GND, DQ, and VDD as shown in Figure 6.

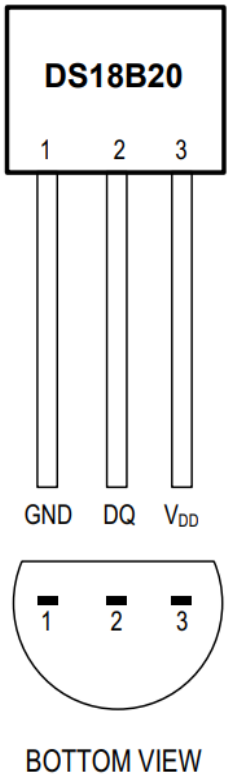

Figure 6. Pin configuration for DS18B20 cropped from its datasheet [11].

- GND is the ground pin.
- DQ is the data pin.
- VDD is the supply voltage pin.

Figure 7 shows the connections for the prototype.

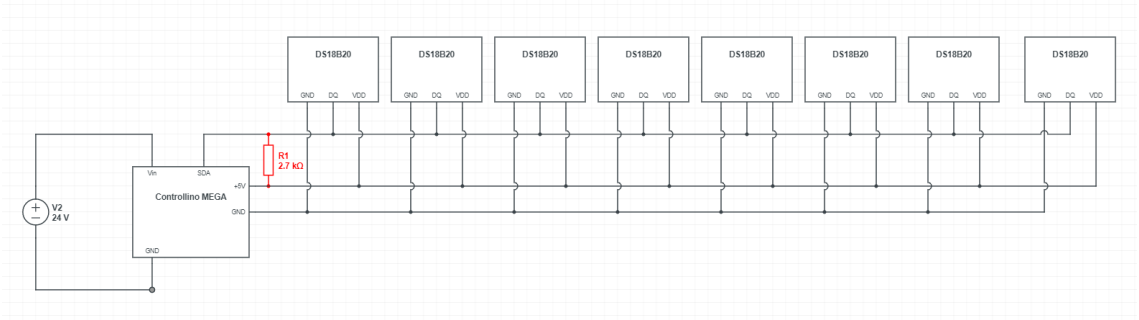

Figure 7. Circuit schematic.

It is recommended in the datasheet that when connecting the DS18B20 sensor to a PLC, a 5 kΩ pullup resistor be connected in parallel with VDD and DQ pins. However, this proved to be causing problems when using more than five sensors in a single bus. This was caused due to insufficient current in the line, so a 2.7 kΩ resistor was chosen instead for a total of eight sensors in the bus.

5.2 Programming of PLC

Before being able to use the Arduino IDE to inject code into the Controllino, an extension package must be added for the IDE to recognize the PLC. Instructions and download of the package, along with the needed library, can be easily found on the Controllino webpage.

The Controllino is programmed to acquire the temperature readings from eight DS18B20 sensors connected in a single bus, create a JavaScript object notation (JSON) string of the temperature data, and publish it to the Mosquitto MQTT broker. The program employs the use of 6 libraries to do this, one made by Arduino, and the rest are community-developed libraries. These are further explained below.

5.2.1 Controllino.h

Made by Controllino. It adds additional aliases for defining the pins of the PLC in the program. The library is used in this program to define the pin to which the temperature sensors are connected to the PLC through the line, as shown in Listing 1.

```
#define ONE_WIRE_BUS CONTROLLINO_PIN_HEADER_SDA
```

Listing 1. Defines the SDA pin header of the PLC in which the sensors are connected as "ONE_WIRE_BUS".

5.2.2 OneWire.h

Made by Paul Stoffregen. It lets you access 1-wire devices made by Maxim/Dallas. It is required for DallasTemperature.h to work. A OneWire object needs to be created using the pin defined in Listing 1.

5.2.3 DallasTemperature.h

Made by Miles Burton. This library uses the OneWire library to add helpful methods and functions to control the DS18B20 and other similar temperature sensors.

The program uses three functions: "getDeviceCount()", "requestTemperatures()", and "getTempByIndex(i)". The first one returns the number of devices connected to the bus, and it is only used for debugging purposes in this program. The second and third are used in conjunction as the former sends the command to the sensors to register a temperature and needs to be called before the latter can parse through each sensor by the given index in the bus to gather the data.

5.2.4 Ethernet.h

Made by Arduino. It allows the Arduino board to connect to the internet through an Ethernet connection. For this library, the media access control (MAC) address and the internet protocol (IP) address need to be declared as parameters. The former is simple to come by as it is provided by Controllino themselves. The latter takes a few more steps, as a program must be run while the Controllino is connected to a network to find out its IP. The program used can be found in Appendix 1.

To tell the server to begin listening for incoming connections, the begin(MAC, IP) function is used.

5.2.5 MQTT.h

Made by Joël Gähwiler. It adds MQTT functionality to the Arduino to publish/subscribe to an MQTT broker. The functions “connect()” and “publish()” are used from this library. They allow to create a connection with the MQTT server and to publish messages, respectively.

While not in use at this stage of the project, the functions “subscribe()” and “onMessage()” can be used to receive incoming data from the server.

5.2.6 ArduinoJson.h

Made by Benoît Blanchon. It adds helpful functions to create/parse JSON objects. A JSON document needs to be created to hold the memory representation of the object using the “StaticJsonDocument” in which a fixed capacity must be specified. The block of code responsible for creating the JSON string is shown in Listing 2.

```
//Prepare sensor data for MQTT transfer
for (int i=0; i<numberOfDevices; i++) {
      String sensorID = "sensor";
      sensorID += i;
      float temperature = sensors.getTempCByIndex(i);
      doc[sensorID] = temperature;
}
char buff[256];
serializeJson(doc, buff);
```

Listing 2. Arduino code used to generate a JSON object with sensor information and creating a string out of it.

This code will loop through each sensor in the line and save the temperature reading in the created JSON document “doc” as the value, with the “sensorID” as its name. The name in this instance is created using the built-in “String” reference. However, this has been known to cause memory related issues and may need to be changed in the future when the scope of the project becomes bigger and as a result, the program more complex. The buff parameter is then used as a buffer for the JSON document to be saved in as a character array, which is then sent using the MQTT library publish() function.

A copy of the full code can be found in Appendix 2.

5.3 Design of SCADA System

Node-Red is the tool used to create the SCADA system for this project. Its design allows the wiring of different devices and services in an intuitive way. For this project, it is used to handle the data it receives from the PLC through the MQTT server to save it in a database, as well as showcase it in a rudimentary HMI. The design of the flow is shown in Figure 8.

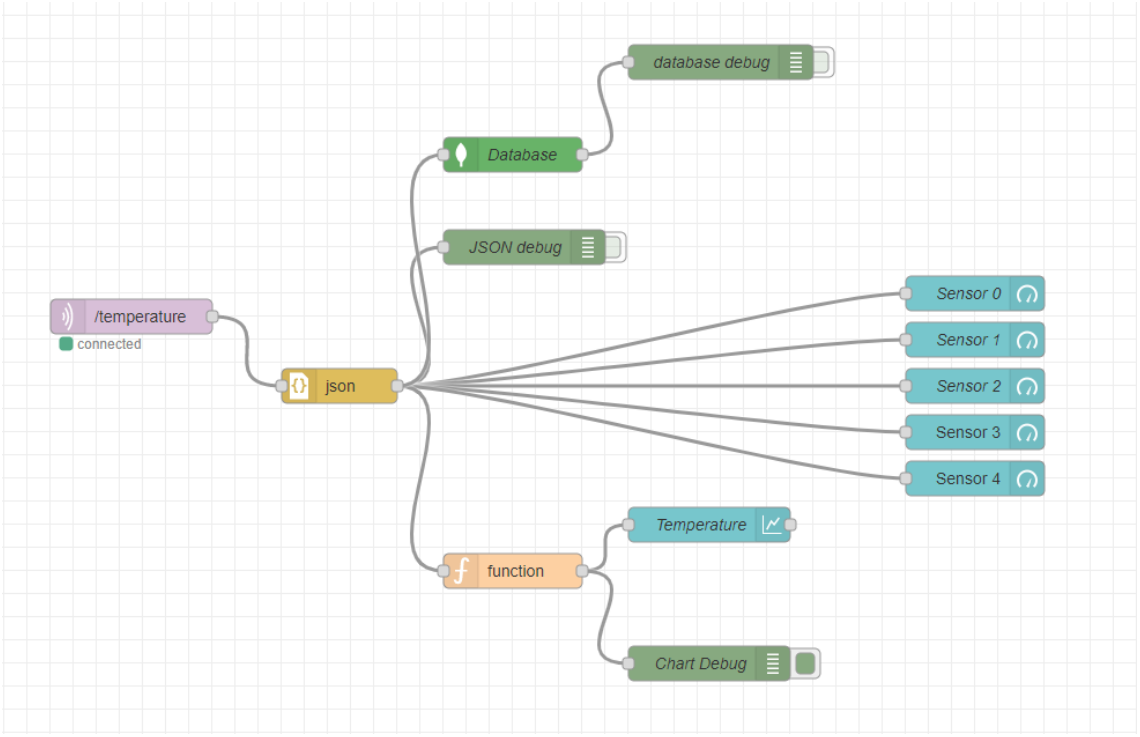

Figure 8. Node-red flow design.

The flow starts with an “MQTT in” node configured to listen at the specified topic for incoming data, which in this case is the JSON string with the sensor data from the PLC. The node will output this data as a string and pass through the “json” node, which converts it into a JSON object. This is necessary to easily parse through the object for the required values.

From here, the flow splits into three sections. The upper section saves the data to a database using “Mongodb3 in” to store the object into the configured MongoDB database. The middle blue nodes are all “gauge” nodes used to create the HMI portion of the system. Each node will retrieve the value of a single specified sensor and show it on an interface as a gauge widget, this interface can be seen in Figure 9.

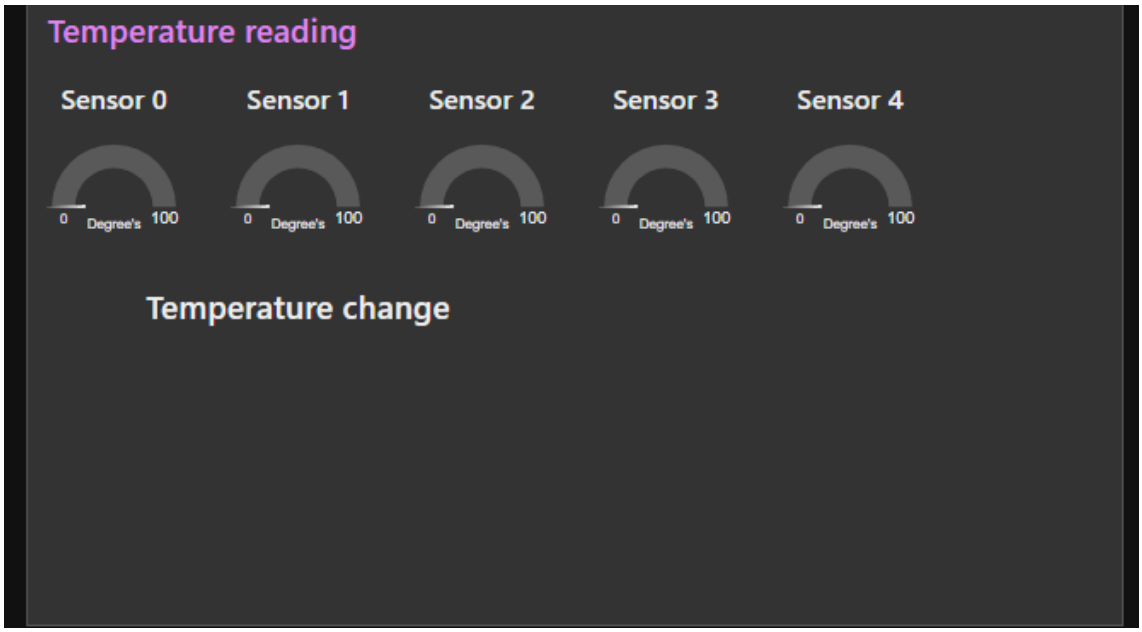

Figure 9. Node-Red interface.

Figure 9 also shows a chart with all the sensor values shown along a time axis. This is created on the bottom portion of the flow, where the JSON object is passed through a “function” node which lets users create a JavaScript function to run against the messages being received by the node. The code used on this node is showcased in Listing 3.

```
var a=msg.payload;
var msg1 = { topic:"Sensor00", payload:a.Sensor00 };
var msg2 = { topic:"Sensor01", payload:a.Sensor01 };
var msg3 = { topic:"Sensor02", payload:a.Sensor02 };
var msg4 = { topic:"Sensor03", payload:a.Sensor03 };
var msg5 = { topic:"Sensor04", payload:a.Sensor04 };
return [ [msg1, msg2, msg3, msg4, msg5] ];
```

Listing 3. JavaScript block of code inside “function” node.

This node receives the JSON object comprised of eight sensor values and extracts the first five, it will the output each value as five different outputs which are needed for the

next node. The “chart” node it is connected to receives input values as the payload which will attempt to convert into a number, this number is then plotted into a chart in the user interface. The number of lines in the chart depends on the number of inputs it receives.

5.4 Instrumentation of Distillation Column.

The column was outfitted with five sensors as shown in Figure 10. One placed in the feed, one at the bottom where the low volatility liquids settle, and one at each stage of the column.

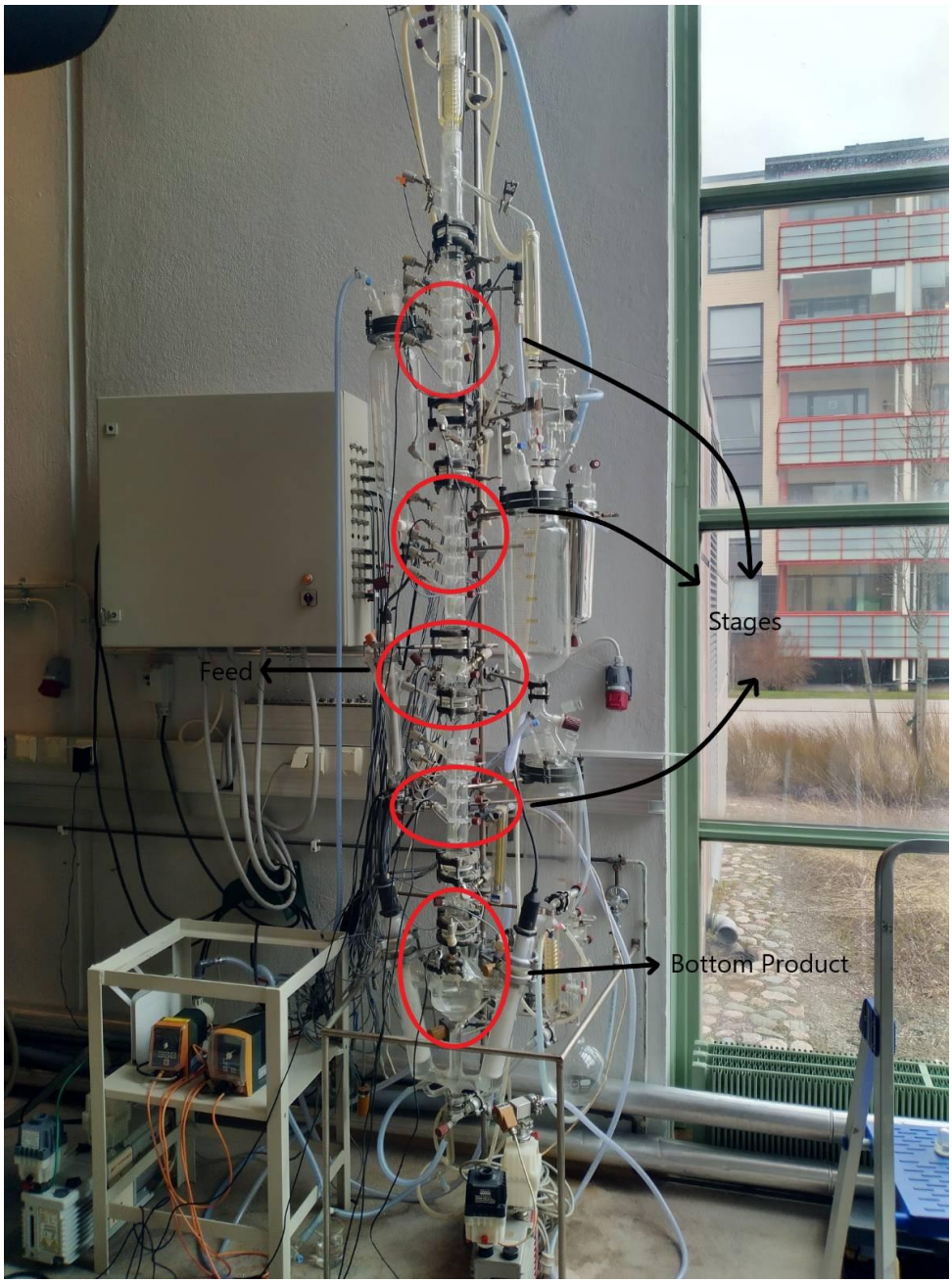

Figure 10. Distillation column outfitted with sensors.

The bottom of the column was filled with water and heated to its boiling point for the initial testing of the prototype. Custom build adaptors needed to be made to place the sensors in the column and create a seal. The liquid inside the column is heated with a rod-type heater for which control is not yet implemented.

6 Discussion

The systems in place worked as designed with no significant trouble. As shown in Figure 11, the PLC successfully generated a JSON string with the temperature data to be sent through MQTT.

Figure 11. JSON string being generated by the PLC.

As it had been previously mentioned, the current method of generating the JSON string employs the usage of the “String” reference which has been known to cause trouble with the already limited amount available of dynamic memory in most PLCs. This method may be something that has to be revised and improved for the later stages of the project when more sensors and control devices are added, limiting the amount of memory available.

The data was properly received and shown in the created interface as illustrated in Figure 12.

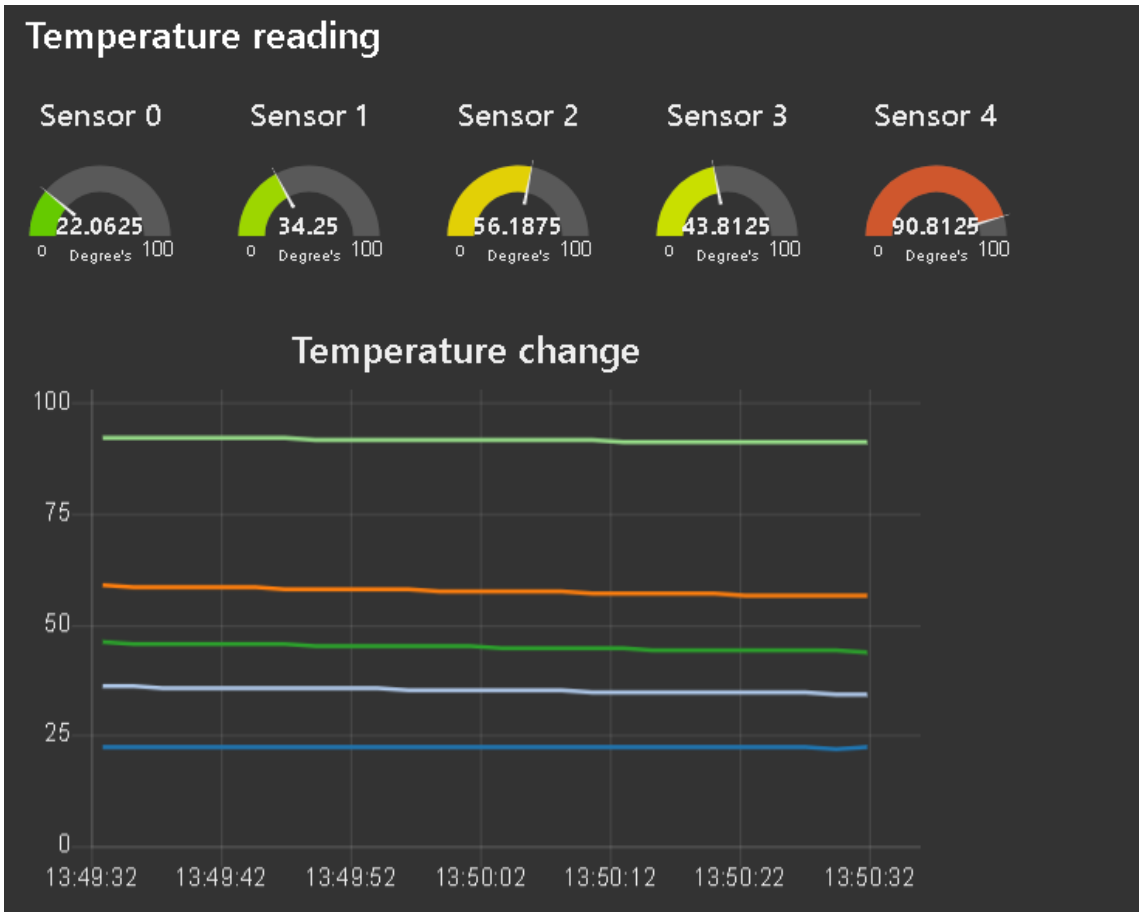

Figure 12. Created HMI displaying temperature data during test run.

This gives a decent general temperature profile throughout the section of the column outfitted with sensors. However, improvements may have to be made to the method in which temperature is displayed on the HMI when considering that the entire column will be outfitted a total of twenty-eight temperature sensors.

How the tools used in this project compare to established proprietary alternatives, such as LabVIEW or services and products provided by companies such as Siemens and ABB, is difficult to judge at this stage. Proprietary technologies are generally more specialized, with a narrower scope to cover than open-source tools. They offer greater stability and tailored support to their users; however, this causes a dependency to be formed and users possess lesser control or flexibility over how their data is handled. Open-source fixes this problem and puts the handling of data in the users' hands, whether this is ultimately more economical is yet to be seen. Situations where one is better can differ

and vary depending on the intended use. In the context of CPS, a finished version of the device must be in place to better compare it to established technologies.

This thesis work begun the process of refurbishing a distillation column with a temperature monitoring system, and due to time constraints, that is where it had to stop. Control of temperature through the implementation of a PID controller is the next recommended step. Eventually, pressure sensing, and control should be made possible through further instrumentation. Further improvements on the HMI and SCADA systems in place is also necessary.

7 Conclusion

The project set out to evaluate open-source technologies as a viable alternative to proprietary ones regarding CPS. A prototype with only temperature measuring functionality and a proof of concept of what would constitute the SCADA systems were successfully developed.

While the project is still in its early stages of development, it is hard to come to a conclusive answer in the prospects of open-source technologies on the industry based on the current results alone. It does show promise as a tool for people who wish to begin a business and do not have the initial funds to spend on proprietary solutions as a low-risk alternative. However, more research into this topic is needed and completion of the device is required before a proper conclusion can be reached.

References

1 Yongxin Liao, Fernando Deschamps, Eduardo de Freitas Rocha Loures & Luiz Felipe Pierin Ramos. “Past, present and future of industry 4.0 – a systematic literature review and research agenda proposal” [online]. 2017. International Journal of Production Research.
URL: http://dx.doi.org/10.1080/00207543.2017.1308576. Accessed (13/04/2021)

2 Wübbeke J, Meissner M, Zenglein MJ, Ives J, Conrad B. “Made in china 2025” [online]. Berlin, Germany: MERICS | Mercator Institute for China Studies; December 2016.
URL: https://kritisches-netzwerk.de/sites/default/files/merics_-_made_in_china_2025_-_the_making_of_a_high-tech_superpower_and_consequences_for_industrial_countries_-_76_seiten_1.pdf. Accessed (13/04/2021)

3 McKinsey Digital. “Industry 4.0 after the hype. Where manufacturers are finding value and how they can best capture it” [Online]. 2016.
URL:https://www.mckinsey.com/~/media/mckinsey/business%20functions/mckinsey%20digital/our%20insights/getting%20the%20most%20out%20of%20industry%204%200/mckinsey_industry_40_2016.ashx. Accessed (04.04.2021)

4 Fisher D, Gould P. "Open-Source Hardware Is a Low-Cost Alternative for Scientific Instrumentation and Research" [online] Modern Instrumentation, Vol. 1 No. 2, 2012, pp. 8-20. doi: 10.4236/mi.2012.12002. Accessed (13/04/2021)

5 Kister HZ. “Distillation -Design-”. New York: McGraw-Holl; 1992.

6 Neutrium. “Distillation Fundamentals” [online]. 2017.
URL: https://neutrium.net/unit-operations/distillation-fundamentals/. Accessed (25/04/2021)

7 Zeid A, Sundaram S, Moghaddam M, Kamarthi S, Marion T “Interoperability in Smart Manufacturing: Research Challenges” [online]. Machines 2019, 7(2):21.
URL: https://doi.org/10.3390/machines7020021. Accessed (11/04/2021).

8 Lee EA. “Cyber Physical Systems: Design Challenges” [online]. EECS Department: University of California, Berkeley; 2008.
URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-8.html (Accessed 11/04/2021)

9 Givehchi, Omid & Jasperneite, Juergen. “Industrial Automation Services as part of the Cloud: First experiences” [online]. ResearchGate; 2013.
URL: https://www.researchgate.net/publication/257402460_Industrial_Automation_Services_as_part_of_the_Cloud_First_experiences. Accessed (21/04/2021)

10 MQTT. “MQTT: The Standard for IoT Messaging” [online]. 2019.
URL: https://mqtt.org/. Accessed (27/04/2021)

11 Maxim Integrated. “DS18B20 – Programmable |Resolution 1-Wure Digital Thermometer” [online]. 2019.
URL: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf. Accessed (27/04/2021)

Code Used to Retrieve IP Data.

```
#include <SPI.h>
#include <Ethernet.h>

// Enter a MAC address for your controller below.
// Newer Ethernet shields have a MAC address printed on a sticker on the
shield
byte mac[] = {
  0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02 };

// Initialize the Ethernet client library
// with the IP address and port of the server
// that you want to connect to (port 80 is default for HTTP):
EthernetClient client;

void setup() {
  // start the serial library:
  Serial.begin(9600);
  // start the Ethernet connection:
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // no point in carrying on, so do nothing forevermore:
    for(;;)
      ;
  }
  // print your local IP address:
  Serial.println(Ethernet.localIP());

}

void loop() {

}
```

Code Used in the Project.

```
#include <Controllino.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <Ethernet.h>
#include <MQTT.h>
#include <ArduinoJson.h>


// Sensor is connected to SDA pin 20 / physical pin 40
#define ONE_WIRE_BUS CONTROLLINO_PIN_HEADER_SDA

//initialize library objects
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
EthernetClient net;
MQTTClient client(256);
StaticJsonDocument<400> doc;


//the media access control address
byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
//ip of the device.
byte ip[] = {192, 168, 8, 101};

int numberOfDevices;



void connect() {

  Serial.print("Connecting...");
  while (!client.connect("controllino", "try", "try")) {
    Serial.print(".");
    delay(1000);
    }

    Serial.println("\nconnected!");

    client.subscribe("/temperature");
}

void messageRecieved(String &topic, String &payload) {
  Serial.print("Incoming message: ");
  Serial.print(topic);
  Serial.print(" - ");
  Serial.print(payload);
}


void setup() {

  Serial.begin(9600);
  Ethernet.begin(mac, ip);
  sensors.begin();
  client.begin("test.mosquitto.org", 1883, net);

  numberOfDevices =  sensors.getDeviceCount();
```

```
  connect();

   // locate devices on the bus
  Serial.print("Locating devices...");
  Serial.print("Found ");
  Serial.print(numberOfDevices, DEC);
  Serial.println(" devices.");


}

void loop() {

   client.loop();

   if (!client.connected()) {
    connect();
   }

   // Send the command to get temperatures
   sensors.requestTemperatures();

   //Prepare sensor data for MQTT transfer
   for (int i=0; i<numberOfDevices; i++) {
    String sensorID = "sensor";
    float temperature = sensors.getTempCByIndex(i);
    sensorID += i;
    doc[sensorID] = temperature;
   }

   serializeJson(doc, Serial);
   Serial.println();

   char buff[256];
   serializeJson(doc, buff);



   //Send data through MQTT
   client.publish("/temperature", buff);



   delay(1000);


}
```