Sergey Morozov, Alissa Subina, Andrei Lebedev, Roey Weinreb

Testbench system for prototyping based on Controllino and Raspberry Pi

Metropolia University of Applied Sciences

Bachelor of Engineering

Electronics, Information Technology

Innovation Project

29 May 2023

Abstract

Authors:	Sergey Morozov[1], Alissa Subina[1], Andrei Lebedev[2], Roey Weinreb[1]
Title:	Innovation Project
Number of Pages:	31 pages + 0 appendices
Date:	29/05/2023
Degree:	Bachelor of Engineering
Degree Programme:	Electronics[1], Information Technology[2]
Supervisors:	Anssi Ikonen (Principal Lecturer)

The aim of this innovation project was to design, assemble and test a universal IoT testbench system based on Controllino PLC and Raspberry Pi, which can be integrated into a hydroponic system. The control panel was intended to be cost-efficient, hence the components were chosen accordingly. The project included multiple sensors, varying from inexpensive sensors for hobbyists (e.g., HC-SR04) to industrial grade probes (e.g, GMP252). Throughout its development, there was a necessity to use communication protocols such as Modbus RTU, 1-Wire, UART and Bluetooth LE (for wireless sensors). At the same time, a web application for displaying all the measured data was created and made ready to work with all the other components. At the assembly stage, the students faced some unexpected problems, which were mostly solved. The values from the sensors were successfully parsed to the website and the corresponding tests were conducted and recorded. The raised problems were considered and will be overcome in the future project, which will be tested with the real plants for UrbanFarmLab of Metropolia UAS.

Keywords: IoT, Farming, Hydroponics, Controllino, RaspberryPi, Sensors, pH, EC, Modbus RTU, 1-Wire, UART, BLE, RuuviTag, Website

Contents

List of Abbreviations

BLE: Bluetooth Low Energy.

EC: Electrical Conductivity.

GPIO: General Purpose I/O.

IoT: Internet of Things.

PLC: Programmable Logic Controller.

RX: Receiver.

RGB: Red Green Blue.

RTU: Remote terminal unit.

SBC: Single Board Computer.

TDS: Total Dissolved Solids.

TX: Transmitter.

UI: User Interface.

UART: Universal Asynchronous Receiver Transmitter.

1 Introduction

A control panel, also known as an automation box, is a key element in most industry sectors, particularly in agriculture. While many commercial control panels for automation of plant harvesting already exist, there is still a lack of low-cost and easily customisable setups often desired by small-scale innovative companies. Indeed, the growing number of indoor farming start-ups that use hydroponics, aquaponics or aeroponics approach need to monitor various environmental parameters and control system devices in a cost-efficient way. Moreover, customisation and remote access of an automation box might significantly reduce both maintenance cost and R&D investments for start-ups making them more viable and competitive on the market.

Fortunately, over the last decade, there has been a rise of open electronics platforms including microcomputers (i.e., Raspberry Pi) and programmable logic controllers (i.e., Controllino) supplied by free and open-source software. While the former is typically used as a server for sending data to a cloud, the latter has industry-level protection of inputs and outputs allowing to operate high-voltage devices and the variety of sensors out of the box. The combination of Raspberry Pi and Controllino might represent a powerful IoT PLC module for a low-cost and customisable automation box which enables online monitoring and controlling of both wired and wireless devices in a small indoor farm.

2 Aim and Objectives

The aim of this innovation project is to design, assemble and test a universal IoT control system based on Controllino PLC and Raspberry Pi which could be used as a backbone for practical applications in agriculture, animal husbandry, or research facilities.

The project implementation entails the following objectives:

1. to research control panel components, sensors, and IoT services available on the market or in open access.
2. to connect wired sensors and output devices (a pump, a fan, an alarm buzzer, and a light strip) to Controllino PLC.
3. to establish communication between wireless sensors and Raspberry Pi.
4. to store raw data on Raspberry Pi and visualize them on an external display.
5. to develop remote monitoring & control using web UI.
6. to build the test rack system where both Controllino and Raspberry Pi including all the sensors and output devices parts are integrated.
7. to test the entire system to observe its reliability, pros, and cons.
8. to write the detailed documentation to make the project repeatable.

3 System Design

Since the control panel was intended to be compact and meanwhile multifunctional, the electronic components had to be selected deliberately. To achieve this, certain factors had to be considered. Firstly, each element of the system had to be studied thoughtfully to ensure that it corresponds the aim of the project. Secondly, the components had to be acquired from a particular source determined by Metropolia UAS i.e., the item availability had to be examined. Overall, the automation box consisted of 8 wired and 2 wireless probes and sensors controlled by Controllino MAXI and Raspberry Pi.

The behaviour and data flow of the system is demonstrated in Figure 1. The start point is represented by a filled black circle, while the circle with a dot inside is an end point. The Controllino PLC field is a main loop of the program activating by powering up the system. Yellow boxes are readings from input devices, green and red ones are ON and OFF states of the output devices, and blue boxes represent processes related to data manipulation. The actual code implementation is built based on this finite state machine (Figure 1).

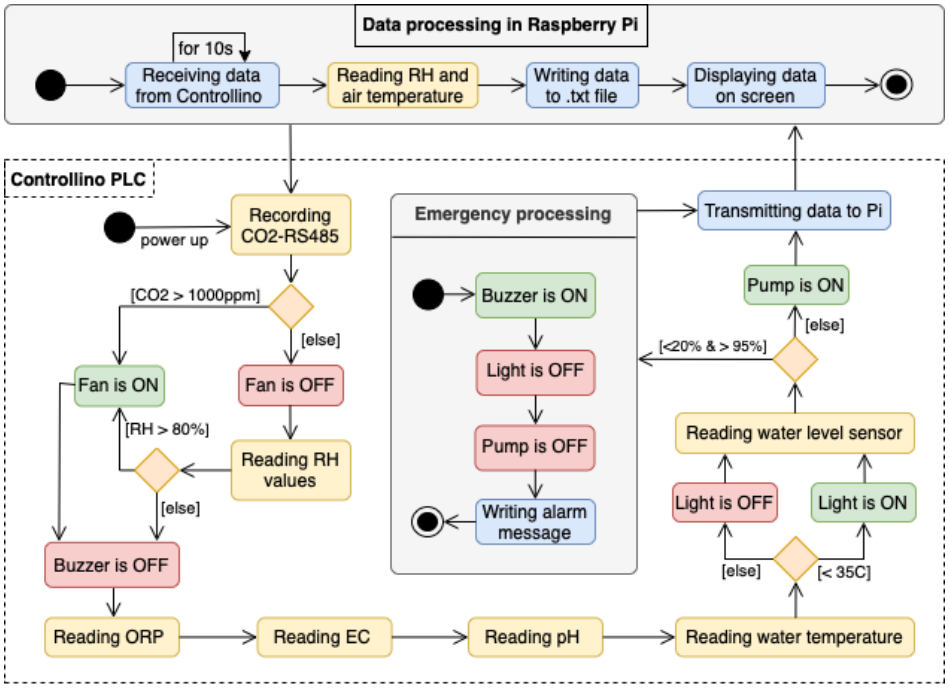

Figure 1. Behaviour diagram of the test system

The diagram shown on Figure 2 provides a detailed overview of the system connections and components. The Controllino MAXI PLC was placed at the center of the schematic as it was connected to a wide range of sensors. The colors of wires refer to the type of communication used; the red color signifies the normal data-pin connection, while purple marks the communication through the protocols. The HPP271 and GMP252 sensors communicate with Modbus RTU protocol and are the only sensors in the system, which operate on 24V. The wires of these sensors are attached to the in-built RS485 module of Controllino MAXI.

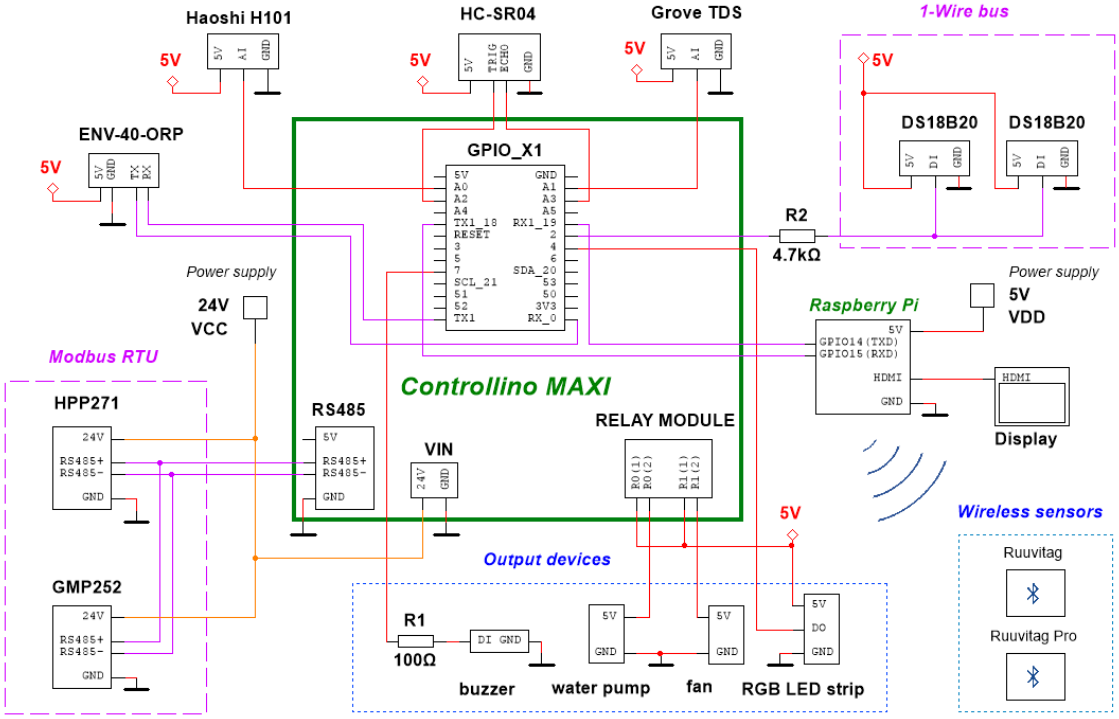

Figure 2. Component diagram (designed in Multisim).

Meanwhile, two DS18B20 sensors use 1-Wire technology for communication with the master device and have only one data line in series with 4.7kΩ pull-up resistor. Another communication method used is UART running via TX and RX pins and implemented with ENV-40-ORP and Raspberry Pi. The Raspberry Pi itself is responsible for the wireless part of the system: it consists out of several RuuviTags, which are communicating to the RaspberryPi via BLE. It also has an LCD display connected to it. Lastly, the output devices, water pump and fan use

the internal relay module, while the buzzer and RGB LED strip use the GPIO X1 pins.

3.1 Controllino PLC

Controllino Maxi is a programmable logic controller (PLC) that is based on the Arduino Mega board with additional embedded hardware modules such as a real-time clock, an Ethernet interface, and a serial to Modbus RTU converter. Despite Controllino is not officially supported by Arduino, it has its own Arduino library ‘CONTROLLINO’ that allows a user to work in Arduino IDE using a USB cable for C/C++ code uploading and serial monitoring. The project implied utilization of the great number of wired components in an efficient and affordable way, hence the corresponding microcontroller was required. The selection was made in favor of Controllino PLC.

3.2 Wired Probes

Various types of sensors and probes were integrated into the test system to demonstrate its versatility for the end user (Table 1). Some of the sensors could be used only in water, while others were designed for air measurements. Protocol communication of Controllino with wired probes also varied from simple end-to-end configuration (digital, analog, UART) to bus systems (1-Wire and Modbus RTU). Both low-budget sensors (e.g., HC-SR04 & DS18B20) and more expensive industrial probes (i.e., Haoshi H101 & Vaisala GMP252) have been presented. The basic principle and actual application of each probe integrated into the system are presented in the following section. Please note, that it is not an extensive probe overview, but rather a summary of the system components (Table 1).

Table 1. Basic description of sensors and probes used in the test system. The sensors are ranked depending on the level of complexity and price.

Product name	**Manufacturer**	**Work principle**	**Physical property**	**Protocol**	**Price (pcs)**
HC-SR04	generic	ultrasonic	water level	digital	≈ 4 €
DS18B20	Dallas Semiconductor	digital output (pre-calibrated)	water temperature	1-Wire	≈ 10 €
Grove TDS	Seedstudio	resistance	EC	analog	≈ 15 €
Haoshi H101	DFRobot	electrochemical	pH	analog	≈ 60 €
ENV-40-ORP	Atlas Scientific	electrochemical	ORP	UART	≈ 100 €
GMP252	Vaisala	optical	CO_2	Modbus	≈ 750 €
HPP271	Vaisala	capacitance	H_2O_2	Modbus	≈ 2300 €

3.2.1 HC-SR04

In hydroponics, plants must be constantly supplied with water. The HC-SR04 was used as a water level meter for the water reservoir. Its working principle is based on reflection of ultrasonic waves from the surface. In other words, the high-frequency sound of 40 kHz is emitted through the trigger pin of the sensor and when it reaches an object the sound rebounds to the echo pin. Finally, the following formula 1 is used to determine the distance [1.]

$$d = \frac{v_{sound} * t}{2} \quad (1)$$

Where ‘d’ is a resulting distance in meters, ‘v’ is the speed of sound in the air in meters per second and ‘t’ is the time interval between sound is being transmitted and received in seconds.

- Input voltage: 5V
- Operating current: 15mA

- Accuracy: 0.3 cm
- Measurement range: 2 – 400 cm

3.2.2 Grove TDS by Seedstudio

The total dissolved solids (TDS) sensors are widely used in agriculture for water quality monitoring. As the name suggests, TDS meter measures the amount of dissolved solids such as minerals, salts, metals, and organic matter in the water usually denoted in parts per million, or ppm [2]. In hydroponics, the levels of TDS may indicate, for instance, the abundance of nutrients or, on the contrary, their deficiency, which typically vary between 600 ppm and 1000 ppm. However, it depends on the type of a plant as some ideally require much higher ppm, such as peppers (1400 – 1700 ppm) and broccoli (1950 – 2500 ppm) [3.]

There are two TDS measurement techniques, gravimetric analysis, and conductivity. The gravimetric method is the most precise, however is time-consuming and generally is more complex as it includes liquid solvent evaporation [4.] The Grove TDS sensor by Seedstudio used in this project is related to the conductivity-based method, providing a response within milliseconds with satisfactory accuracy. The waterproof probe allows TDS to be measured in the range of 0 – 1000 ppm, which is first given as a raw value, then converted to a voltage and finally inserted into TDS formula 2 shown below. Where "v_{EC}" is the input value in volts and coefficients in front of it are the part of the cubic polynomial equation.

$$\mathrm{TDS} = (133.42 * (v_{EC})^3 - 255.86 * (v_{EC})^2 + 857.29 * v_{EC}) * 0.5 \quad (2)$$

In addition to TDS, water quality can also be assessed by measuring electrical conductivity, or EC. Since the sensor can detect dissolved ionized solids, which are indicative of electrical conductivity in water, it can also operate as an EC meter [4]. In comparison to TDS, which indicates the amount of nutrients in the

water, EC evaluates the ability of plants to absorb the nutrients [3]. As in this project the preference was given to EC measurements the corresponding conversion (formula 3) was performed.

$$TDS = \frac{EC}{2} \quad (3)$$

In which TDS is still calculated in ppm and EC in uS/cm.

- Input voltage: 3.3V / 5V
- Operating current: 3 – 6mA
- Accuracy: ± 10% of full scale (25 °C)
- Measurement range: 0 – 1000 ppm

3.2.3 Industrial pH Sensor

As EC, pH readings are equally important for successful plants growing. Acidity of the water affects the amount of nutrients absorbed by plants. The unregulated pH levels can lead to toxicity of some elements or lack thereof. The acceptable value usually varies between 5.5 pH and 6.5 pH, while 0 pH is considered acidic, 7 pH is neutral and 14 pH is alkaline [5.]

The Haoshi H101 by DFRobot is an industrial pH meter, a pH electrode of which is manufactured from the sensitive glass membrane. This advanced sensor ensures rapid and consistent response times, making it an ideal choice for its intended purpose. The working principle of it is based on the formula 4 below.

$$pH = 3.5 * v + offset \quad (4)$$

Where “v” is the voltage in volts calculated from the raw analogue value coming from the sensor and “offset” is a compensation for the deviation [6.]

- Input voltage: 5V
- Operating current: not specified
- Accuracy: ± 0.1pH (25 °C)
- Measurement range: 0 – 14 pH

3.2.4 ORP Probe

Oxidation/reduction potential or ORP measurement is an additional method used to enhance the effect of nutrients. Although not as popular as EC and pH, ORP can provide important information such as the presence of harmful bacteria in a solution. All this is due to the oxidation and reduction process i.e., loss and gain of electrons in the water. The higher the ORP values the higher the amount of oxygen in the liquid and vice versa. In hydroponics the ideal ORP range is 300 – 500 mV [7.]

The Atlas Scientific ENV-40-ORP is a laboratory grade probe for monitoring ORP changes in a variety of applications. Its top part is manufactured from an unreactive metal – platinum, which makes it possible to study the electron activity without disturbing the environment. The probe is connected to an embedded ORP circuit, EZO-ORP, which operates with either UART or I^2C interfaces. This allows the user to send commands such as start calibration, change the baud rate, switch to low power mode and much more [8.]

- Input voltage: 3.3V / 5V
- Operating current: 13.3 – 14.5mA / 13.8 – 18.3 mA
- Accuracy: ± 1mV
- Measurement range: -2000 to 2000mV

3.2.5 Temperature Sensor DS18B20

Another significant factor of any water-based growing system is the water temperature. Temperature affects many biological processes from photosynthesis to germination. Cold water, for instance, can reduce the metabolic rate of plants, while hot water can cause root rot. Therefore, the ideal temperature range should vary between 18 °C and 25 °C, depending on the type of plant being grown [9.] Apart from this, temperature has an impact on the measurement accuracy of the EC and pH sensors described previously. In order to avoid deviations in the output, temperature compensation is necessary.

The digital thermometer DS18B20 included in the project operates via 1-Wire interface by Dallas Semiconductors. This means that several devices can operate on one data bus. For this reason, two temperature sensors were integrated into the system so that the difference in measurements could be observed.

- Input voltage: 3.0V / 5V
- Operating current: 1 – 1.5 mA
- Accuracy: ±0.5°C (-10°C to +85 °C)
- Measurement range: -55°C to +125 °C

3.2.6 Vaisala GMP252

GMP252 represents a CO_2 probe designed for industrial applications. The probe is pre-calibrated on a factory and has a minimum drift over long period of time. It provides accurate reading in comparison with abovementioned sensors and has a wide measurement range from 0 to 10000 ppmCO_2. The CO_2 reading are based on optical sensor and CARBOCAP® technology. Temperature compensation is calculate automatically based on readings from an embedded temperature sensor. Both digital and analog data of outputs are available. The former is implemented over Modbus RTU protocol allowing to connect GMP252 probes to bus networks controlled by PLC. The latter is based on current (4-

20mA) and voltage (0-10V) outputs. The device has a M12 male for easy and reliable plugging-in and plugging-out [10.]

3.2.7 Vaisala HPP271

HPP271 is the other industrial probe similar to GMP252 in terms of quality and protocols communication supporting analog output and Modbus-RTU. Instead of CO_2, it measures H_2O_2, humidity and temperature and calculates its derivative physical characteristics. It is primarily installed in bio-decontamination. The basis of H_2O_2 measurement is rooted in the comparison of readings from two composite humidity sensors using PEROXCAP. All of the electronics is encapsulated into metallic enclosure to increase the probe lifetime and endurance [11.]

3.3 Output Devices

The output devices add functionality to the system and ensure that it works properly. The project included following output devices:

- Water pump, 5 – 12V.
- Fan, 5 – 12V.
- RGB LED strip, 5V.
- Buzzer, 5V.

The water pumps in hydroponics are responsible for water flow. The system needs at least one water pump to circulate the water and gradually deliver nutrients to the roots. Obviously, the larger the system the more pumps it requires. Moreover, the hydroponic systems often have a fan. The fan creates air circulation, allowing the gardener to control such parameters as humidity and temperature of the environment. The RGB LED strip can be used for plant lighting, as it can be programmed to emit almost any coloured light, which positively affects growth. The light colour can be tailored to the specific needs of specific plants. The buzzer alarm is an optional feature, however, it is beneficial to include, because it can prevent serious problems e.g., water tank leaks, high

pH levels. If certain values deviate from the norm, the buzzer will produce a loud sound, and a warning will be sent to the main page of the website, which will be described in the following chapters.

3.4 Raspberry Pi

Raspberry Pi is a small single board computer (SBC) that has most of the standard PC's features, such as internet connectivity, operating system, etc. Because it is a low cost and capable device it can be used in a vast range of fields such as: weather monitoring, robotics, home automation, educational projects, DIY projects, and many more.

In the scope of the project, the RaspberryPi had to be setup for work with special settings. These will be covered step by step.

1. sudo raspi-config -> Interface Options -> Serial Port -> Would you like the login shell to be accessible over serial? Choose 'No' -> Would you like the serial port hardware to be enabled? Choose 'Yes' -> Reboot
2. nano /boot/config.txt -> use Ctrl+W to search for 'uart' -> find the 'enable_uart' line and make sure that the value is set to 1
3. ls /dev/tty* -> make sure that the list has a value /dev/ttyS0
4. Done! The device is ready to work with GPIO.

3.5 Wireless Sensors

To monitor the growing environment, wireless sensors are very useful and easy to implement. Ruuvi sensors were chosen for the project, specifically RuuviTag and RuuviTag Pro. The sensors are connected via Bluetooth to the Raspberry Pi and are very accurate, reliable, and cost efficient. RuuviTag is a wireless Bluetooth sensor node that measures temperature, air humidity, air pressure and movement. Live measurements and history data can be accessed directly on a smartphone using Ruuvi's mobile app for Android and iOS. Typical

absolute accuracy ±0,2 °C at temperature range of 5-60 °C, output resolution 0,01 °C.

RuuviTag Pro is a rugged IP certified environmental sensor for temperature, humidity, and acceleration monitoring. Is tolerates wet and rough conditions, and it is suitable for indoor and outdoor use. Measurement resolution is 0,01 °C, while absolute accuracy is ±0,1 °C across the temperature range of –20 to +50 °C, with no calibration. In the scope of the project, these sensors will be used with a custom Python script, which will be described below.

```
sensor = RuuviTag(mac)
sensor2 = RuuviTag(mac2)

conn=create_connection('ruuvidata.sqlite')

while True:

    data = sensor.update()
    data2 = sensor2.update()

    line_sen = str.format('Sensor - {0}', mac)
    line_tem = str.format('Temperature: {0} C', data['temperature'])
    line_hum = str.format('Humidity:    {0}', data['humidity'])
    line_pre = str.format('Pressure:    {0}', data['pressure'])

    line_sen2 = str.format('Sensor - {0}', mac2)
    line_tem2 = str.format('Temperature: {0} C', data2['temperature'])
    line_hum2 = str.format('Humidity:    {0}', data2['humidity'])
    line_pre2 = str.format('Pressure:    {0}', data2['pressure'])


    with conn:
        msr=(mac, data['temperature'], data['humidity'], str(datetime.now().strftime("%d-%m-%Y %H:%M:%S")))
        msr2=(mac2, data2['temperature'], data2['humidity'], str(datetime.now().strftime("%d-%m-%Y %H:%M:%S")))

        create_measurement(conn, msr)
        create_measurementPro(conn, msr2)

    # Wait for 9 seconds and start over again
    try:
        time.sleep(1)

    except KeyboardInterrupt:
        # When Ctrl+C is pressed execution of the while loop is stopped
        print('Exit')
        break
```

Figure 3. Snippet of the Python script.

The Python script shown in Figure 3 has objects that contain connection information for database access. After this, come two functions that do the actual logging into the wireless sensors' database, one for each RuuviTag.

4 Web Application

Node.js was used as the web page server. The web application itself was done using HTML, EJS, CSS, JavaScript. The database which holds user credentials was done in SQLite3. A previously mentioned Python script was used to interact with wireless sensors.

Frameworks used for development were Bootstrap for frontend CSS and Express.js for backend JavaScript. Originally, TailwindCSS was used for frontend, but Bootstrap was chosen over it due to installation issues. Among the backend Node.js Package Manager packages, the most significant were:

- bcrypt, a password hashing tool;
- passport.js, a user authentication middleware;
- body-parser, a node.js body parsing middleware;
- knex, an SQL query builder;
- ReadlineParser, a package that translates machine code into human-readable text;
- Serialport, a package that allows opening connections with serial ports from the server's backend.

4.1 Port Listening Functionality

Before the user starts using the application, the web page server starts listening to the RaspberryPi's 'ttyS0' port, as shown on Figure 4. If the port cannot be accessed, an error message will be displayed in the node.js' console. Otherwise, all of the data coming to the said port is piped into the ReadlineParser's instance, then it is assigned to the valuesCon (values from Controllino) variable. To trigger Controllino's response, a string is sent to it. If the string cannot be sent, an error message will be displayed in the node.js console.

```
const port = new SerialPort({
    path:'/dev/ttyS0', //try connecting via pins
    baudRate: 19200,
    autoOpen: false
})

const parser = new ReadlineParser()

port.open((err) => {
    if (err) {
      console.error('Error opening port:', err.message)
    } else {
      console.log('Serial port opened.')
      port.pipe(parser)
      // Read data from the serial port
      parser.on('data', (line) => {
        valuesCon = line
      })
      const sendData = 'Hello, Controllino!'
      port.write(sendData, (err) => {
        if (err) {
          console.error('Error writing to port:', err)
        } else {
          console.log('Sent:', sendData)
        }
      })
    }
  })
```

Figure 4. Port listening function.

To trigger Controllino's response, a string is sent to it. If the string cannot be sent, an error message will be displayed in the node.js console.

4.2 REST API

The web application routing was done using a REST API. Routes will be covered one by one below.

4.2.1 Page access functionality

In some routes, the functions 'checkAuthenticated' and 'checkNotAuthenticated' are used. Their purposes are described with reference to Figure 5:

```
function checkAuthenticated(req, res, next) {
    if (req.isAuthenticated()) {
        return next()
    }

    res.redirect('/login')
}

function checkNotAuthenticated(req, res, next) {
    if (req.isAuthenticated()) {
        return res.redirect('/')
    }
    next()
}
```

Figure 5. Page access functions.

- checkAuthenticated: this function will not let the user access the page until they are successfully logged in.
- checkNotAuthenticated: this function will not let the user access the page if the user is already logged in; pages that use this function can only be accessed if the user has not logged in yet.

4.2.2 REST API routes

Starting route. Referring to Figure 6, this route is accessed when the user types the website's URL into the browser. Afterwards, they are immediately redirected to the login page.

```
app.get('/', checkNotAuthenticated,(req, res) =>{
    res.redirect('/login')
})
```

Figure 6. Starting route.

Login page. Figure 7 illustrates, that when the user arrives at the page, code from the GET method pulls all of the registered users' credentials from the database into an array and renders the page. In the POST method, credentials entered by the user are authenticated against the credentials array when the user clicks the 'Sign in' button on the screen.

```
app.get('/login', checkNotAuthenticated, async (req,res) =>{
    users = await knex.getAll(knex.usersdb, 'userCredentials')
    res.render('login.ejs')
})

app.post('/login', checkNotAuthenticated, passport.authenticate('local', {
    successRedirect: '/home',
    failureRedirect: '/login',
    failureFlash: true
}))
```

Figure 7. Login page routes.

If authentication is successful, the user is redirected to the home page, otherwise the page refreshes, prompting the user to try different credentials.

Registration page. Originally, the user was supposed to have the ability to create an account from the Login page, which would lead them to the registration page. As of right now, the Registration page cannot be accessed from the Login page. However, if the user desires to register an account, they can manually access the page by entering the /register URL to the browser, thus triggering the page's GET method, which is shown in Figure 8 together with the POST method. The page has very minimal UI, but the functionality is complete: the user will be greeted with a form asking their name, email address, and password.

```
app.get('/register', (req,res) =>{
    res.render('register.ejs')
})

app.post('/register', async (req,res) => {
    try {
        const hashedPassword = await bcrypt.hash(req.body.password, 10)
        await knex.createThingy(knex.usersdb, 'userCredentials',{
            id: Date.now().toString(),
            name: req.body.name,
            email: req.body.email,
            password: hashedPassword
        })
        res.redirect('/login')
    } catch {
        res.redirect('/register')
    }
})
```

Figure 8. Registration page routes.

After registration, the user is redirected to the Login page, so that they can enter the website. Once the user is done filling the form and clicks ‘Register’, the POST method is triggered, saving the new user credentials to the SQLite database, encrypting the user password in the process.

Home page. After the user logs into their account, the home page is displayed.

```
app.get('/home', checkAuthenticated, (req, res) =>{
    res.render('index.ejs')
})
```

Figure 9. Home page route.

The home page is the main page of the website.

fetchAPI (‘values’). Figure 10 provides visual representation of the fetchAPI backend part, which gives data for the frontend to display.

```
app.get('/values', async (req, res) => {
    let ruuvi1Q = await knex.ruuvidata('ruuvi1').max('time').select('Temp', 'Hum')
    let ruuviProQ = await knex.ruuvidata('ruuviPro').max('time').select('Temp', 'Hum')
    let ruuviValues = ruuvi1Q[0].Temp + '&' + ruuviProQ[0].Temp +
    '&' + ruuvi1Q[0].Hum + '&' + ruuviProQ[0].Hum + '&'
    let values = ruuviValues.toString();
    if (valuesCon !== undefined) {
    values += valuesCon.toString();
    }
    console.log(values)
    const response = {
        value: values
    }

    res.json(response)
})
```

Figure 10. fetchAPI route.

When the user logs in and the home page is displayed, the fetchAPI route pulls wireless sensor data from their database. After that, the wireless sensor data and the wired sensor data are combined into a string of values, which is later sent to the frontend to be processed and displayed.

4.3 Knex Functions

Knex functions were made to make SQL queries against databases using JavaScript, which allows to avoid using raw SQL queries that clutter the code and add complexity. They are kept in a separate file to completely avoid messing with the web server's code. When the web server boots up, the file is imported. Some functions are unused but are kept to be used in later versions of the project.

The file consists of three parts: database connections, functions, and exports. All of them will be described.

4.3.1 Database connections

As shown in Figure 11, the database connections are objects that contain information which when paired with functions allows them (functions) to be executed on said databases.

```
const usersdb = knex({
    client:'sqlite3',
    connection: {
        filename: 'db.sqlite3'
    }
})

const measureddb = knex({
    client:'sqlite3',
    connection: {
        filename: 'msrdb.sqlite3'
    }
})

const ruuvidata = knex({
    client:'sqlite3',
    connection: {
        filename: 'ruuvidata.sqlite'
    }
})
```

Figure 11. Knex database connections.

4.3.2 Database functions

As previously mentioned, functions when paired with database connections allow execution of different pre-coded queries on the selected databases. These functions are depicted in Figure 12.

```
function createThingy(db, table, thingy){
    return db(table).insert(thingy)
}

function getThingy(db, table, id){
    return db(table).where('id', id).select()
}

function getAll(db, table){
    return db(table).select('*')
}

function deleteThingy(db, table, id){
    return db(table).where('id', id).del()
}

function updateThingy(db, table, id, thingy){
    return db(table).where('id', id).update(thingy)
}
```

Figure 12. Knex functions, which perform the most frequently used SQL queries.

4.3.3 Database function file exports

The exports are a list, which contains the objects and functions that can be used outside the file with the functions, when said file is imported. The list can be seen in Figure 13.

```
module.exports = {
    measureddb,
    usersdb,
    ruuvidata,
    createThingy,
    getThingy,
    getAll,
    deleteThingy,
    updateThingy
}
```

Figure 13. Knex function file exports.

4.4 Home Page Frontend Functionality

Referring to Figure 14, the home page has a part of the fetchAPI functionality implemented in its frontend. The second part of the fetchAPI starts immediately after the first part ends: the data string sent to the frontend is sliced into separate variables, each of which are then assigned to an HTML element to be displayed. The fetchAPI refreshes the displayed values every 50 milliseconds, or as soon as the Controllino sends new values.

```
fetch('/values')
  .then(response => {
    if (!response.ok) {
      throw new Error('Error fetching data')
    }
    return response.json()
  })
  .then(data => {
    const wlvl = document.getElementById('wlvl')
    const ph = document.getElementById('ph')
    const h2o2 = document.getElementById('h2o2')
    const co = document.getElementById('co')
    const wtemp1 = document.getElementById('wtemp1')
    const ec = document.getElementById('EC')
    const atemp1 = document.getElementById('atemp1')
    const rh1 = document.getElementById('rh1')
    const wtemp2 = document.getElementById('wtemp2')
    const orp = document.getElementById('ORP')
    const atemp2 = document.getElementById('atemp2')
    const rh2 = document.getElementById('rh2')
    const alerts = document.getElementById('alerts')

    console.log(data.value)
    const arr = data.value.split('&')
    console.log(arr[0])

    if (wlvl) {
      wlvl.textContent = arr[4]
    }
    if (ph) {
      ph.textContent = arr[5]
    }
    if (h2o2) {
      h2o2.textContent = arr[6]
```

Figure 14. Snippet of the frontend' fetchAPI code.

Additionally, the frontend has status functionality: it displays different messages when the water level is too low, too high, or when there is no data coming from Controllino. This can be observed in Figure 15.

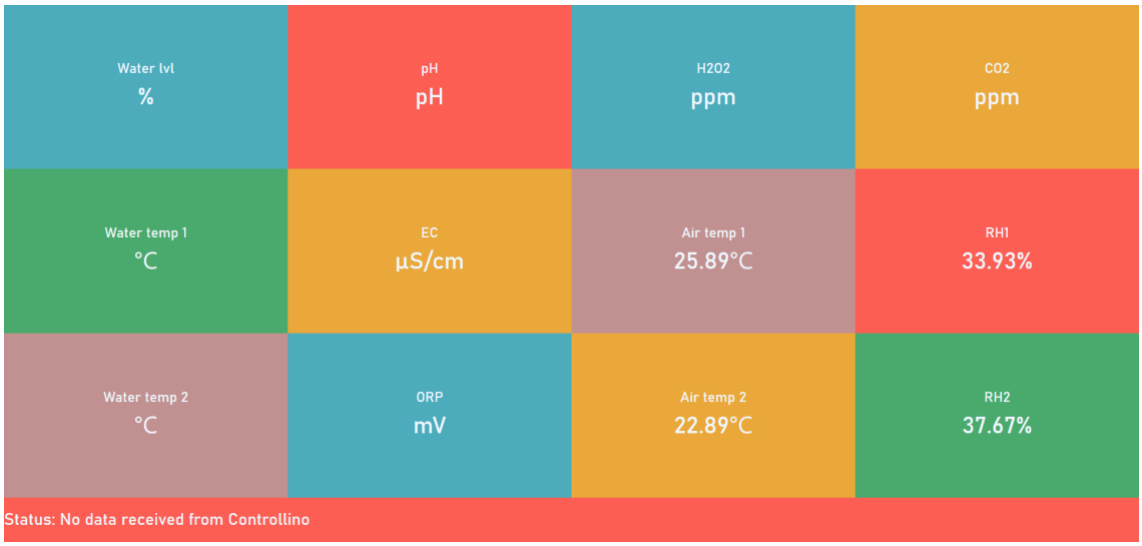

Figure 15. Demonstration of status functionality; in this case, only the wireless sensor data is available.

5 Results and Discussion

5.1 Prototyping and Installation

The actual prototyping and assembling of this system were conducted in AIoT Garage and Electronics Laboratory of Metropolia UAS, respectively. The initial design was implemented on a large prototyping breadboard with three voltage sources connectors: 5V, 12V, and 24V respectively (Figure 16).

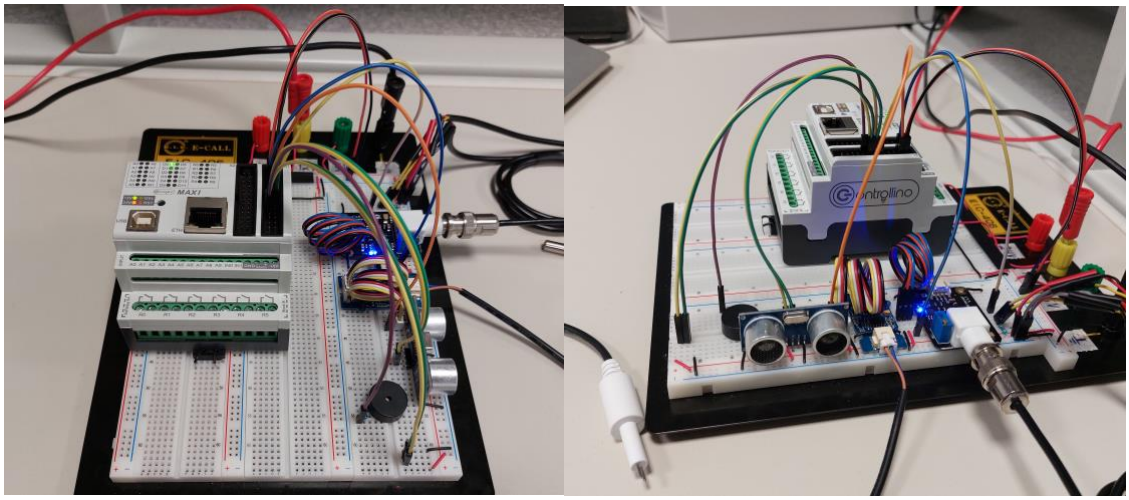

Figure 16. Photos of the prototype: front and side views.

PLC was also embedded on a breadboard using small cable wires as well as sensor meters. However, unwrapping of the setup becomes more time consuming with adding new components to the system. Thus, when most of the components of the system have been tested separately, it was decided to build a sandwich-type panel with two levels (Figure XX). The size of each plate was 32 x 24 cm. The bottom plate was used for storing and hiding all the cables going from the meters to actual probes. Six improvised shafts were inserted into the bottom plate to hold up the top one, where all the components were mounted using M4 bolts and nuts. In addition, a 25 cm DIN rail was attached to the top plate for holding a small breadboard, Controllino PLC, Raspberry Pi with its DIN case (Figure 17).

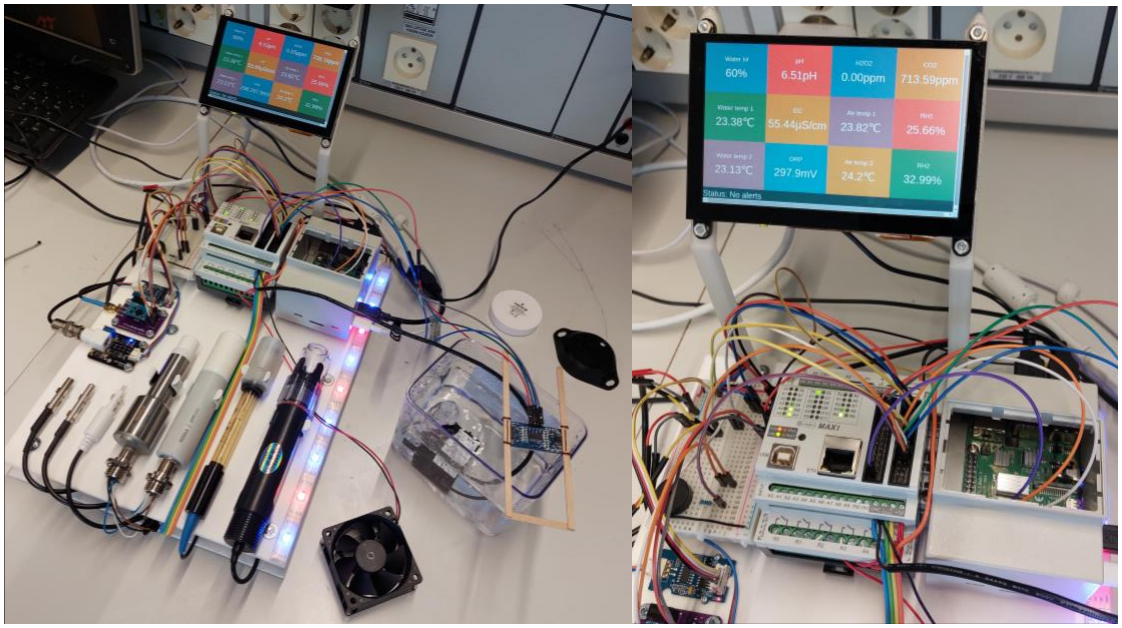

Figure 17. Sandwich-type panel of the testbench system and UI.

The stands for an external screen attached above the DIN rail were printed out in Automation Lab at Metropolia UAS using name of the 3D printer Ultimaker 2 Extended +. The buzzer was inserted in the breadboard and the LED strip was glued to the side of the plate, while HC-SR04, a fan and a pump located apart the panel. The rest of the sensors and meters were attached to the top plate either using bolts or C-shape probe holders. In this configuration, the test system was easy to store and setup (Figure 17).

5.2 Sensor Calibration

As mentioned earlier some of the sensors are temperature dependent. The temperature compensation for EC can be easily achieved from the linear equation 5.

$$EC = [1 + a * (t - 25)] * EC_{25} \quad (5)$$

Where a = 0.020 °C^{-1}, which is a compensation factor, “t” is the current temperature, “EC_{25}” is an uncorrelated value. The reference temperature is 25°C since the ratio of EC/EC_{25} is always 1 [12.]

As part of the system's preparation, the pH and ORP sensors had to be calibrated to reflect a true value. The calibration of Haoshi H101 required two-point calibration method, which means that the sensor had to be immersed in two different standard solutions. Firstly, the sensor was checked at 7.00 pH and the readings were displayed on the serial monitor of Arduino INO. The difference between the solution value and the real value was calculated and used as the "Offset" in the formula 4 mentioned above. Finally, the sensor was immersed in 4.00 pH and left for a short time until the output value was correct and stable. The calibration process of ENV-40-ORP was different due to built-in calibration protocol, which reduces the setup time considerably. The ORP was immersed in one calibration solution of 225mV and when the values had settled, the command "cal,225" was sent, the sensor saved the settings, and the calibration was done.

5.3 Test of the System

The actual test design was implemented based on state conditions of a finite state machine (Figure 1). The scenarios were listed in the Table 2 with a simple pass/fail results and optional comments. Such method allowed a user to test every component of a system and its behaviour without reassembling the whole setup. Isolating components/devices under test in the state machine modifying its code significantly speeded up the testing process. This testing approach was actively used during a prototyping phase which simplifies and standardizes the troubleshooting process.

Table 2. The test suite experiment with the system

Test case	Assertion	Results	Comments
Blow towards the GMP252 probe	Fan should turn ON	PASSED	-
Bring a drop of H_2O_2 closer to the HPP271 probe (but don't touch)	Fan should turn ON	PASSED	-

Immerse ORP probe into the tap water	ORP value should be 100-400 mV	FAILED	UART produces errors at 24V
Immerse pH probe into the tap water	pH value should between 6-8	PASSED	-
Immerse EC probe into the tap water	EC value should be 0-400 uS	PASSED	-
Immerse temperature probe into the tap water	Temperature values should be 15-30°C	PASSED	-
Immerse temperature probe into the hot water	LED strip shall switch to OFF mode	PASSED	-
Hold a hand close to water level sensor	LED strip and pump shall turn OFF, alarm is ON	PASSED	-
Connect display to Rasp Pi and run shell script	Web UI shall work and update values within 10s interval.	PASSED	-
Run the system over two hours in continuous mode	No issues should be observed	PASSED	-

5.4 Observed issues

Throughout the project implementation several issues were identified. Most of them were related to sensors. First, many Modbus devices required at least 12V voltage source on the power line (e.g. Vaisala GMP252, Figure X). However, the Vaisala HPP271 probe had the voltage threshold of 15V, thus either separate powering up of Modbus communication should be applied or Controllino PLC must be powered by 24V. Surprisingly, the latter solution also caused issues with the UART communication (see below).

Secondly, the original EC sensor by DFRobot was not waterproof, hence it had to be replaced with a waterproof TDS sensor produced by Seedstudio. The third issue also came from DFRobot production, where its pH meter and firmware were not able to compensate pH values for temperature. The next generation of

pH meters by DFRobot fixed this issue, but they were out of stock in Finland. Since temperature compensation for pH has non-linear nature depending on the type of pH sensor and meter, and investigation of a compensation formula based on empirical data was out of scope for this project, this issue remained unresolved, and an easy solution would be to buy the next generation pH meter by DFRobot or from other vendors.

I^2C bus protocol should be supported by Controllino, but the sensors and meters, which worked well with Arduino Uno and Mega, were not able to communicate over I^2C. Moreover, similar issues were observed on the Internet and the seller company did not provide any solution for it. Fortunately, the ORP sensor used in the project had a UART mode. Overall, Controllino offers four available UART ports, two of them were typically busy with serial communication with PC and/or Modbus RTU devices (Serial & Serial3, respectively), so eventually one of the ports was reserved for ORP.

In turn, serial communication (UART) of ORP sensor failed when powering Controllino Mega to 24V. While Serial1 did not work at all, Serial2 and SofwareSerial had unstable readings with random errors in the messages. Lowering the voltage level to 15V helped resolve this issue. However, long-term consequences of this solution are unknown, because 11.5 – 13.8V is the recommended supply voltage for 12V input, therefore potential overheating and/or logic problems might be expected. This issue was also mentioned by other Controllino users [13].

Limited Modbus RTU implementation was a problem in the beginning of the project because none of the official Arduino libraries worked with Controllino. The library recommended by Controllino ‘ModbusRtu.h’ is severely outdated and is not supported by Arduino. It was slow and had cluttered code that was difficult to read, modify and maintain. After investing considerable time, the library started working in the state machine.

Communication between Raspberry Pi and Controllino was an issue during website development, due to Controllino being unable to communicate when using USB-A to USB-B cable. After thorough investigation and observation of possible workarounds, it was decided to switch to communication utilizing the GPIO pins. For this to work, the RaspberryPi’s Serial Interface settings had to be changed, and Controllino had to be forced into using a specific Serial port. After all this, the communication started working in expected ways, and the website development continued.

6 Conclusions and further work

After long development, and several issues that forced looking for completely different approaches, the goal was achieved, and the system worked as originally intended. Most of the problems encountered were eliminated. The documentation was made as easy to understand as possible, so that the system can be reproduced by other people. The project has a future as a monitoring system in the UrbanFarmLab for practical application in vertical farming. The next version of the project will feature different sensors and will receive an overhaul to the website and code optimization.

References

1 Rui Santos. Complete Guide for Ultrasonic Sensor HC-SR04 with Arduino [Internet]. Porto, Portugal: Random Nerd Tutorials; Date unknown. URL: https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/. Accessed 25 May 2023

2 Terry Freeman. Total Dissolved Solids(TDS) [Internet]. Edmonton, Alberta: Corrosionpedia; 20 April 2019. URL: https://www.corrosionpedia.com/definition/1103/total-dissolved-solids-tdsBarbara. Accessed 25 May 2023

3 Jamie. Hydroponics TDS Level | Why It Matters & How To Adjust [Internet]. United States: WhyFarmit; 2023. URL: https://whyfarmit.com/hydroponics-tds-level/. Accessed 25 May 2023

4 Multiple Authors. Total dissolved solids [Internet]. United states: Wikipedia; 23 May 2023. URL: https://en.wikipedia.org/wiki/Total_dissolved_solids. Accessed 25 May 2023.

5 Chris. Understanding & Maintaining pH in Hydroponics [Internet]. United States: Happy Hydro Farm; 29 November 2020. URL: https://happyhydrofarm.com/understanding-ph-in-hydroponics/. Accessed 26 May 2023.

6 DFRobot. Gravity: Analog pH Sensor / Meter Pro Kit for Arduino [Internet]. Shanghai, China: DFRobot; Date unknown. URL: https://www.dfrobot.com/product-1110.html. Accessed 26 May 2023.

7 Agrowtronics. Oxidation Reduction Potential (ORP) [Internet]. Zephyr Cove, United States: Agrowtronics; 2021. URL: https://www.agrowtronics.com/oxidation-reduction-potential-orp/. Accessed 26 May 2023.

8 Atlas Scientific. EZO™ ORP Circuit Datasheet [Internet]. Long Island City, United States: Atlas Scientific; October 2021. URL: https://files.atlas-scientific.com/ORP_EZO_Datasheet.pdf. Accessed 26 May 2023.

9 Laurence Brad. What is the Ideal Water Temperature for Hydroponics? [Internet]. Nir Am, Israel: Growee; Date Unknown. URL: https://getgrowee.com/ideal-water-temperature-for-hydroponics/ Accessed 26 May 2023.

10 Vaisala Oy. GMP252 User Guide [Internet]. Vantaa, Finland: Vaisala Oy; March 2018. URL: https://docs.vaisala.com/r/M211897EN-D/en-US/GUID-A317ABA4-

7D33-4F40-8488-6AA3088D8AC5/GUID-9810E120-4E3F-485C-84F6-5E0D97938DCB Accessed 29 May 2023.

11 Vaisala Oy. User Guide Vaisala PEROXCAP® Hydrogen Peroxide Probe HPP271 [Internet]. Vantaa, Finland: Vaisala Oy; March 2020. URL: https://www.vaisala.com/sites/default/files/documents/HPP271-User-Guide-in-English-M211888EN.pdf Accessed 29 May 2023.

12 Aqion. Hydrochemistry & Water Analysis [Internet]. Germany: Aqion; 25 April 2018. URL: https://www.aqion.de/site/112. Accessed 26 May 2023

13 JamesC. Issues with MEGA Serial1 [Internet]. Austria: Controllino Community; 10 May 2023. URL: https://community.controllino.com/forums/discussion/issues-with-mega-serial1/ Accessed 29 May 2023.